VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Yifeng Jiang

ONLINE BANK MANAGEMENT SYSTEM

Technology and Communication
2020

ACKNOWLEDGEMENTS

I cannot appreciate more for last 4 years' study in Finland, and Vaasa university of Applied sciences. I believe it will bring me a lot to my future career.

I must express my gratitude to my supervisor, Dr. Moghadampour Ghodrat. Without his patient guidance. I cannot make this happening. I also want to thanks Dr.Seppo Mäkinen, he gave me lots of help and advises about my thesis during the hard time in pandemic.

Finally, thanks for my parents for their support.

Yifeng Jiang

21.11.2020

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology

ABSTRACT

Author	Yifeng Jiang
Title	Online Bank Management System
Year	2020
Language	English
Pages	55
Name of Supervisor	Ghodrat Moghadampour

The objective of this thesis was to develop an online bank management system for bank customers and administrator to manage bank accounts. This application can simulate real -world bank activities. It allows bank customers to make virtual bank transactions, view bank transactions, manage their accounts and send messages to the administrators. For the bank administrators, this application allows the managing of bank customers' account, viewing bank history, adding virtual money to a certain account and replying bank customers' messages.

Through the optimized combination of functional modules to achieve various management details, the management process will achieve the best degree of automation, and error rate in the accounts will be minimized.

The application was developed by using the ASP.NET technology and C# as the main programming language. All the bank data was stored in the Azure SQL database and retrieved for display when needed. Other technologies such as JavaScript (jQuery), CSS and HTML (CSHTML) were also used.

For the customer best convenience, this application is going to be a web application. The webpages can adjust automatically to different devices according to the current screen size. The application has tested with different web browsers, mobile devices and PCs to make sure it is user-friendly to all the customers.

Keywords ASP.NET, C#, Azure SQL, JavaScript, bank management

CONTENTS

LIST OF FIGURES AND TABLES

LIST OF CODE SNIPPETS

LIST OF ABBREVIATIONS

ASP	Active Server Pages
SQL	Structured Query Language
ERD	Entity-relationship Diagram
ER	Entity Relationship
URL	Uniform Resource Locater
API	Application Programming Interface
B/S	Browser/Server
VS	Visual Studio
HTML	Hyper Text Markup Language
PDC	Microsoft Professional Developers Forum
TCP	Transmission Control Protocol
API	Application Programming Interface
GUI	Graphical User Interfaces
QR	Quick Response
RDBMS	Relational Database Management System
XML	Extensible Mark-up Language
CLI	Common Language Infrastructure
SHA	Secure Hash Algorithm
HTTPS	Hyper Text Transfer Protocol over Secure Socket Layer
CSS	Cascading Style Sheets

HTTP	Hyper Text Transfer Protocol
SSL	Secure Socket Layer
TSL	Transport Layer Security
IP	Internet Protocol

1 INTRODUCTION

With the development of technology, the internet is everywhere in our life. Data is shared and communicated through PC and mobile devices; everything is more digitalized with society moving forward. So should be the bank service. People have larger demand to use bank services in a more convenient way. The aim of the thesis was to make an online bank management to help people to make life easier.

The characteristics of bank management are a relatively large amount of information processing, a large variety of types being managed, a large number of transfer orders, much related information, and different ways of querying and statistics. To build an application to deal with data listed above is a challenging task which requires the implementation of different aspects in a bank management system.

This application will imitate a real-world bank management system. The goal of this project is to deal with a large amount of customer in a reliable way.

The objective of this project is to build an asp.net web application, which interacts data with Azure SQL cloud database, and the web-page is working with HTTPS.

This project aims are to make the transferred business management clear and visual. Through the optimized combination of functional modules to achieve different management details, the management process will complete a the most significant degree of automation, and the account error rate will be minimized.
For the customers, several benefits will be gained:

- It is more convenient for organizations and individuals to perform bank operations. Efficient transfer operations and resource management for the financial management of the enterprise is enabled.
- It is more convenient for organizations and individuals to view the transfer information statistics. The standardized information statistics interface can provide customers with comprehensive and intuitive personal historical statistics information.

2 TECHNOLOGY BACKGROUND

This chapter explains the programming language, database, tools and technologies used in this project.
This ASP.NET web application was built with Visual Studio 2019 environment, and using .NET Framework (version 4.6.1).
The languages used in this application included C#, HTML and JavaScript. Other technology included Azure SQL database, QR code, and SHA-512.

2.1 C Sharp

C Sharp (also called C#) is an object-oriented programming language. It enables programmers to write applications quickly based on the MICROSOFT .NET platform. MICROSOFT .NET provides a variety of tools and services to maximize the development and utilization of computing and communications. /1/
C # is a safe, stable, simple, elegant, object-oriented programming language derived from C and C ++. It inherits the powerful functions of C and C ++ while removing some of their complex features.
C was designed for the common language infrastructure (CLI). The CLI consists of executable code and runtime environment, which allows the use of various high-level languages on different platforms and architectures.

2.2 HTML

Hypertext Mark-up Language (abbreviated as HTML) is a language used on web pages. Basically, current browsers can read HTML, and use HTML to edit and design web pages. All the supported methods of the HTML language, such as tables, forms, pictures, text, links, programs can also be added to the web page.

C SHARP HTML (abbreviated as CSHML) is the extension files of HTML that contains C# code and it is used for client.

2.3 JavaScript

JavaScript (often abbreviated as JS) is a lightweight, interpretive, object-oriented, first-class functional language. JavaScript is easy to learn and powerful, so it is commonly used in web pages. JavaScript is not only a process-oriented language but also an object-oriented language. In JavaScript, objects are created by attaching methods and attributes to empty objects at runtime, as opposed to the common syntax used in compilation languages to define classes. JavaScript scripts are usually embedded in HTML to achieve their own functions.

2.4 Introduction of ASP.NET

ASP. NET is a part of .NET framework. Not only it is the next important version of Active Server Page (ASP), but also a set of unified specifications has been developed in ASP. NET Web development model, such as enterprise-level Web application services. It also offers a programming model and structure. The generated web application has better stability and better protection than ASP. The ASP.NET has the following advantages: the written code is clear, reusable, and good shareability. This not only allows programmers to develop the required Web applications more easily, but also satisfies the strategic goal of transferring C/S structure applications to Web applications. /2/

2.4.1 Features of ASP.NET

The features of ASP.NET includes the following items:

- ASP.NET provides stable performance, excellent upgradeability, faster development, easier management and network services. The theme throughout ASP.NET is that the website does most of the little work for customers.
- Brand new structure: The new ASP.NET introduces an entirely new concept of managed code (Managed Code), which traverses the entire Windows development platform.
- High efficiency: For a program, speed is very essential. To simplify the program code as much as possible in ASP, so that the system must be ported to a low-performance component. ASP.NET can adequately solve this problem.

- Easy to control: ASP.NET contains "Data-Bounds" (data constraints), which means it will be connected to the data source and will automatically load data, making the control work simple and easy. Language support: ASP.NET supports compiled languages. It runs faster than these compiled languages and is more suitable for writing large applications.
- Better upgradeability: The rapid development of distributed applications also requires faster, more modular, more comfortable operation, more platform support and more reusable development. New technology is needed to adapt to different websites. Network applications and websites need to provide a more robust and scalable service. ASP.NET can adjust to the above requirements.

2.4.2 .NET Framework

Microsoft.NET Framework (commonly known as .NET Framework) is a new order code programming model for Windows. It combines powerful functionality with new technologies to build applications with visually compelling user experiences, to communicate seamlessly across technology boundaries, and to support a variety of business processes.

2.4.3 ASP.NET MVC Framework

ASP.NET MVC Framework is based on the MVC (Model-View-Controller) architecture, so that each model of the application can run under the MVC framework.

- View: Responsible for displaying data and user interface. Under the ASP.NET MVC framework, View can support REST-style URLs.
- Model: Responsible for defining data storage.
- Controller: Responsible for handling the connection between View and Model.

2.5 Bootstrap

Bootstrap is the world's most popular front-end component library. Customize responsive web design makes it mobile-first.

Bootstrap is an open source toolkit consisting of HTML, CSS and JavaScript. Sass variables and mix-ins, responsive grid system and powerful jQuery-based plugins help to organise the ideas. /3/

2.6 Database

The database is a "warehouse that organizes, stores, and manages data in accordance with the data structure". It is a collection of organized, shared, and uniformly managed large amounts of data that has been stored on a computer for a long time.

2.6.1 Cloud Database

A cloud database refers to a database optimized or deployed in a virtual computing environment and can realize the advantages of pay-as-you-go and storage consolidation. According to the type of database, it is divided into the non-relational databases and relational databases generally. The benefits of moving your database to the cloud included a few features:

- Scalability: The cloud database can be quickly, cheaply and efficiently expanded.
- Real -time backup: In cloud database cold backup data is stored for 5 days, and it can be backed up at any time within 3 days to ensure online data security. If a database is self-built, a backup strategy needs to be devised and manually restored. The technical requirements are relatively high.
- Cluster: The cloud database provides the master-slave synchronization function and provides read-only cases, to ensure that some queries with low Real-time performance are completed on the read-only cases, sharing the read pressure of the master database, and failure of the master database. It automatically switches to the slave library to achieve the purpose of disaster recovery; With self-built databases, the master-slave synchronization cluster is needed and a disaster recovery strategy needs to be deployed.
- Monitoring: The cloud database has comprehensive, all-weather three-dimensional monitoring, and fault early warning. Self-built databases need a Database monitoring system.

2.6.2 Microsoft Azure SQL Database

The Microsoft Azure SQL Database is a relational database service based on Microsoft SQL Server and built on the Microsoft Azure cloud operating system to perform Cloud Computing. SQL databases provide divinable performance at several service levels, scale,

and data protection. With these capabilities, customers can focus on how to quickly develop applications and save their time. /4/

2.6.3 ADO.NET

ADO. NET is a cross-era improvement to Microsoft ActiveX Data Objects (ADO), which provides a platform for interoperability and scalable data access. Since data is transmitted in XML format, applications that can read the XML format can process data. In fact, the component that accepts data is not necessarily an ADO. NET component. Previously, when accessing a database, the connection to the database was maintained until the data was retrieved. This way of accessing the database is called connected data access technology. In addition to provide this technology, ADO. NET provides a "disconnected" solution to emulate a database in memory, compared with previous data access technologies. A database object in memory is called a DataSet, and a DataSet can contain multiple tables (DataTable) and views (DataView), allowing relationships between tables (DataRelation), as well as rows (DataRow) and columns (DataColurnn) in a table (Datalable) or view (DataView).

2.7 B/S Structure

In the Browser-Server (Browser / Server) structure. The browser interacts with the Web server, and the webserver interacts with the back-end database, which can efficiently work on different platforms; the server can use high-performance PC, and install databases. The B/S structure simplifies the work of the client. However, the server-side work is heavy under this structure, which requires higher performance of the server.

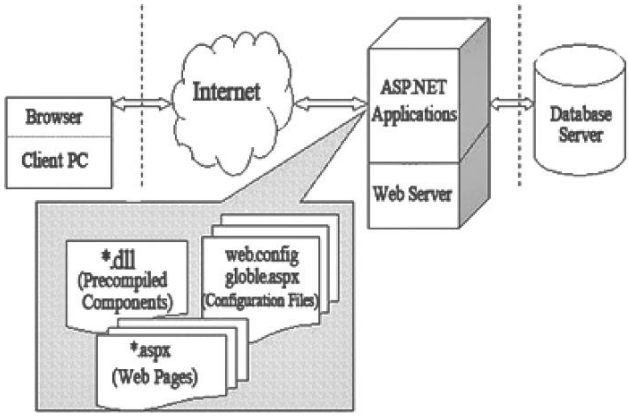


Figure 1. Structure of 3-tier B/S model /5/

2.8 QR Code

A Quick Response code (abbreviate as QR code) is a type of 2-dimensional bar code, invented by DENSO WAVE Company of Japan in 1994. QR codes use four standardized encoding modes (number, alphanumeric, byte (binary), and Japanese (Shift_ JIS)) to store data. They are widely used in mobile device reading and scanning operations all over the world.

2.9 Encrypted Password

In development, a large amount of the user's bank data and personal information is dealt with. From a bank's perspective, the biggest challenge is how to make sure that customers' bank account is in 100% secure. It is often seen that the user account database is frequently hacked, so some measures must be taken to protect the user password, in order to avoid unnecessary data disclosure.

2.9.1 SHA

Secure Hash Algorithm (SHA) is a family of cryptographic hash functions. An algorithm that computes a fixed-length string to which a digital message corresponds, and if the messages entered are different, they have a high chance of corresponding to different strings

SHA-512 is very close to Sha-256 except that it used 1024 bits "blocks", and accept as input a 2^128 bits maximum length string. SHA-512 also has others algorithmic modifications in comparison with Sha-256. /6/

2.10 HTTPS

HTTPS (full name: Hyper Text Transfer Protocol over Secure Socket Layer) is a HTTP channel aimed at security. The security of the transmission process is guaranteed by transmission encryption and authentication based on HTTP. That means adding a security layer under the HTTP page, SSL /TSL. They encrypt the network connection between transport layer and application layer. This system provides authentication and encrypted communication methods. It is now widely used for secure and sensitive communications, such as transaction payments.

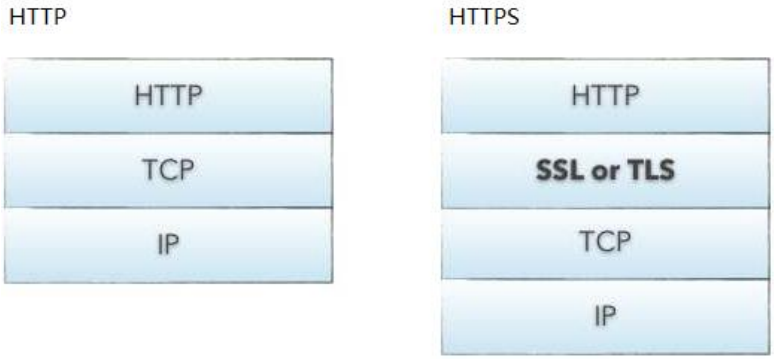


Figure 2. HTTP and HTTPS

3 APPLICATION DESCRIPTION

The online bank management application is described as several different modules and requirements. All the design is analyzed in this chapter.

3.1 Overview Structure

First, on the login page, the user can choose between an administrator and normal bank user. The administrators have the right to view all bank users' information and able to edit their information, even delete their accounts. The administrator users are also able to create a new account.

Second, for the bank customers, after logging in, the user will enter the main dialogue module, which includes "Personal Centre" (modify the user's personal information and view account), "Transfer money", "Withdraw" and "Help" (Send message to administrators), basically meeting the basic needs of ordinary customers in the bank.

3.2 Application Main Modules

This online bank management web-based application has two different user modules: Administrator and bank user module. In the administrator module, administrators can view all bank users' personal information, delete their bank account, deposit to a specific account, view all the bank transactions history, approve register request and reply bank users' messages. In the bank user module, bank users can register a new account on the login page, request a new account for users who already have accounts, ask the administrators to delete an account, transfer funds to another account, edit personal information, view transaction history and send messages to the administrator.

3.3 Security Modules

In an online bank management application, high index modules are indispensable. In the login modules, to maintain the password safety, all the password string stored in the database will transfer to a SHA-512 hash value and QR authentication is also used for login module for second security factor. For the account security, time-out session and block browsers' forward/backward sessions are also made in the security modules.

3.4 Requirements Specification

The requirements specification creates a basis for the acquisition, why and what needs the addition will fulfil. In the requirement specification all the required functions are specified. 1=Must have (Key features which this project must have), 2=Should have (Improved features which make this project better), 3=Nice-to-have (Extra features with enough time and resources).

3.4.1 Administrator Requirements

The table below shows the requirements for the administrator module.

Table 1. Administrator requirements specification

Required features	Description	Priority
Log In	Log in with administrators' nam e and password	1
View bank users' information	View all details for the bank use rs	1
Create account	This enables system to generate a bank account to customer	1
View transaction	This enables administrators to c heck bank users' trading record	1
Delete/Block account	Delete or block a bank account	2
View transaction by conditional query	This allows customer check tran saction by calendar	2
Message function	Reply message to users	2
Deposit	This enables deposit functions	2
Approve account	This allows administrators to ma nage account request	3

3.4.2 Bank Customer Requirements

The table below shows the requirements for the bank customer module.

Table 2. Bank customer requirements specification

Required features	Description	Priority
Log In	Log in with registered username and password	1
Register	Register personal information into the system	1
Apply for a new account	This allows bank user to send new account request to administrators	1
View bank account information	This allows bank user to view all his account information	1
Transfer funds	This enables transfer funds to another account in the system	1
View transaction	This allows users to check their trading history	2
View transaction by conditional query	This allows users search trading records by date and trading type	2
Message function	Reply message to users	2
Withdraw	This allows user to withdraw money from their account	3

3.4.3 Security Requirements

The table below shows the requirements for the bank user module.

Table 3. Security requirements specification

Required features	Description	Priority
Encrypted password	All the password stored in the system should be encrypted	1
QR code authentication	Double security factor	1
HTTPS	Better security and data transmission	1
Time out session	If the user does not do any operations in a specific time, it will automatically return to the login page	2
Browser security	Disabled browser's backward and forward button	2

3.5 Use-Case Diagram

The use-case diagrams are used to describe the relationship between actor and use-case. A use-case diagram contains several elements, such roles, systems and use-cases, and describes various relationships between these elements.

3.5.1 Administrator User-case Diagram

The figure below is the administrator use-case diagram. It describes how the administrator interacts with the application and shows all the actions the administrators can perform. The administrator has the right to view information of all bank users, delete bank account, handle messages, requests, and deposit money to a certain account.

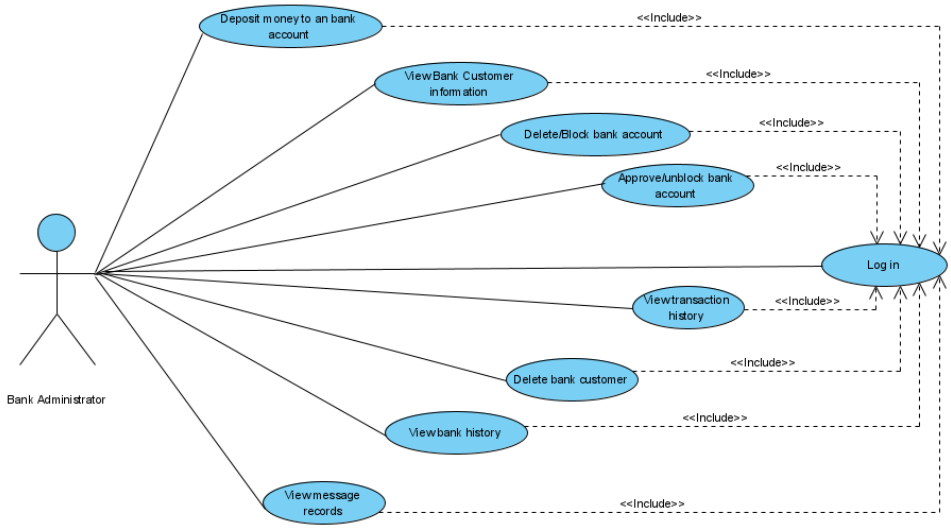


Figure 3. Bank administrator user-case diagram

3.5.2 Bank Customer User-Case Diagram

The figure below shows the bank customer use-case diagram. It presents how the bank customer deals with the application and also shows all the actions that user can perform on this application. A bank user can maintain several bank accounts, updating personal information, change the password, view transactions, transfer money to another account, withdraw cash, send request and messages to the administrator.

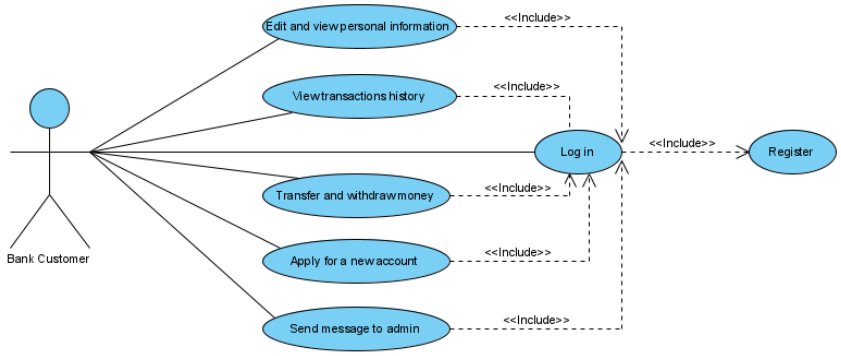


Figure 4. Bank customer use-case diagram

3.6 Sequence Diagram

A sequence diagram simply depicts interaction between objects in sequential order i.e. the order in which these interactions occur. We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by business-men and software developers to document and understand requirements for new and existing systems. /7/

3.6.1 Authentication Sequence Diagram

This sequence diagram shows the processes of how a bank user log into the bank service. First, account number and password are given. If data is correct, then the QR code dialog will pop-up, scan the QR code by the mobile device and get a 4-digital code. If it is all correct, it will redirect to the user homepage, Otherwise it will pop-up a log-in error message.

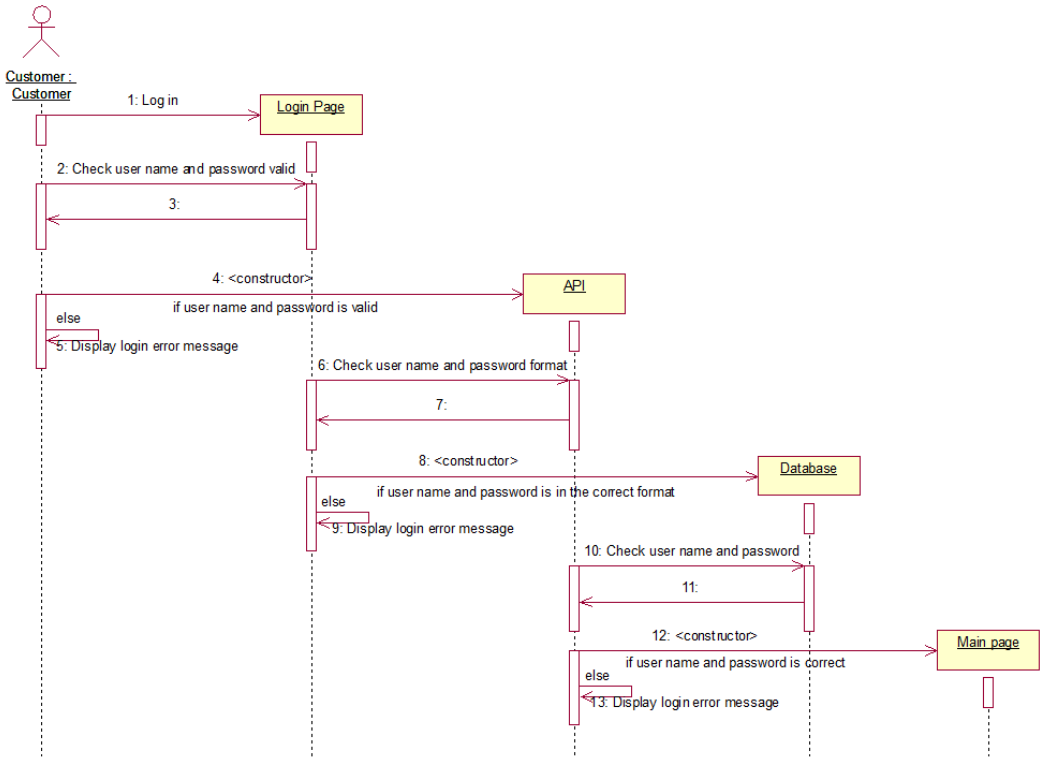


Figure 5. Authentication Sequence Diagram

3.6.2 Trading Sequence Diagram

Bank users can make transactions such as transfer and withdraw. The transaction process is shown below.

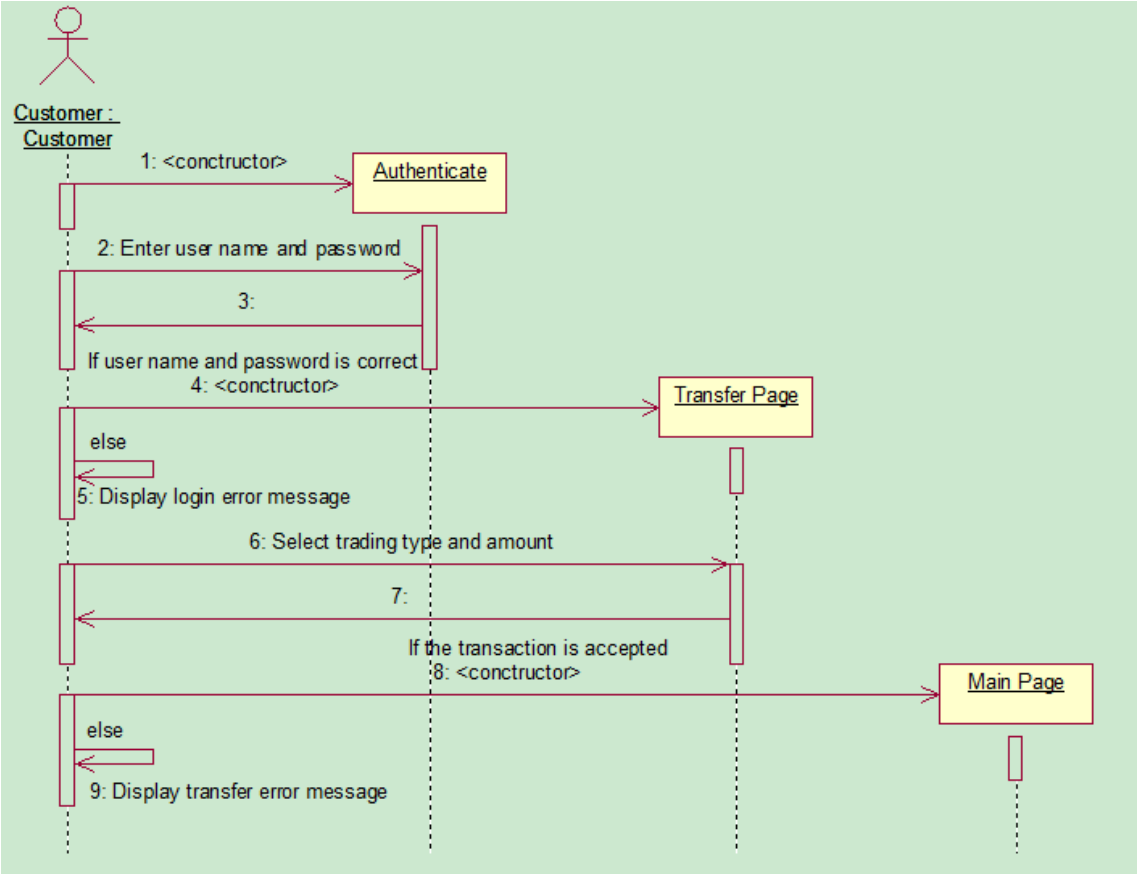


Figure 6. Trading sequence diagram

3.6.3 View Account Transaction Sequence Diagram

In the view transaction diagram, the first step the database validates the user, and after the verification, users can view transactions. Furthermore, the user can also search specific transaction records by choosing trading time period and trading type.

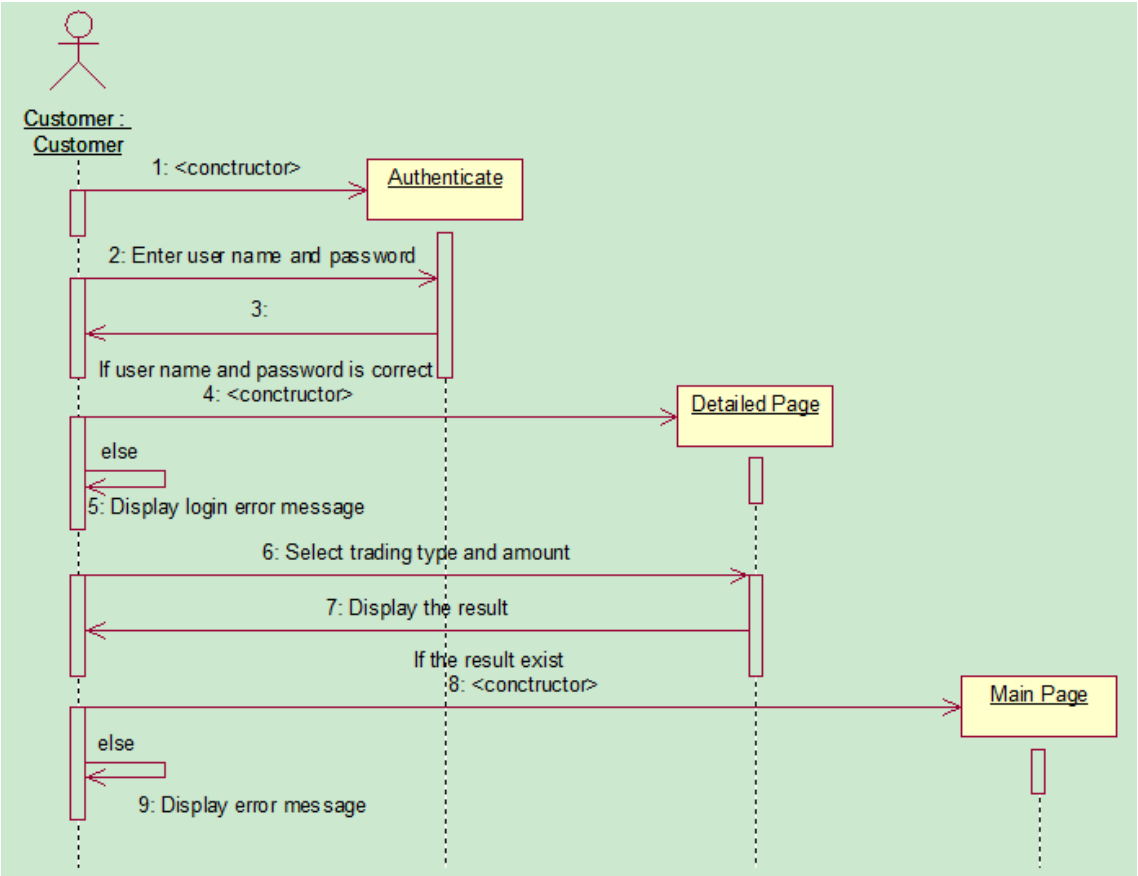


Figure 7. View account transaction sequence diagram

3.6.4 Message Sequence Diagram

The picture below shows the entire process of message function. The logic is the same for both the administrators and the customers. The message information will be directly stored in the database. Users can view the reply information through the message page.

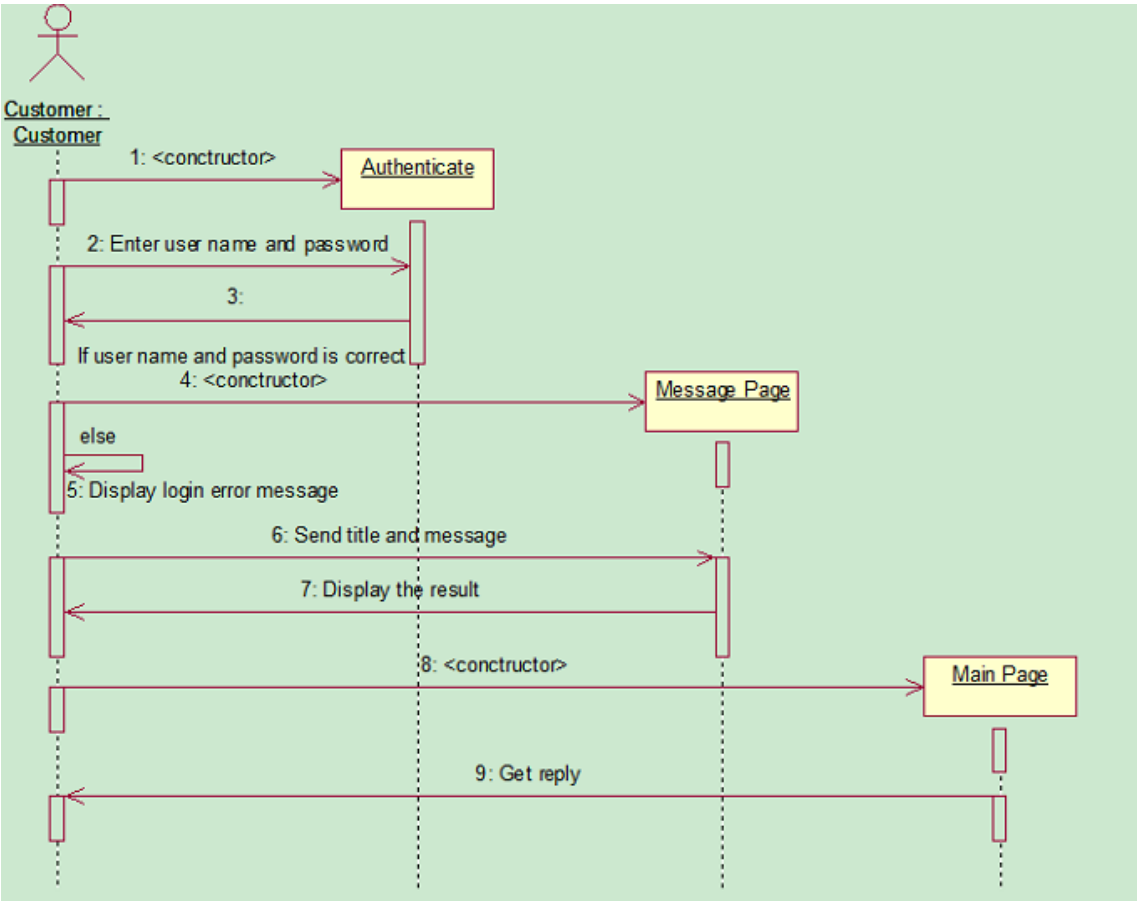


Figure 8. Message sequence diagram

4 DATABASE AND GUI DESIGN

4.1 Database

The database design of the system follows the system function analysis, and according to the objectives of the system requirements. The online bank management system is deployed by the ASP.NET framework and Azure SQL Server storing the data tables.

4.2 Database Connection

On the server-side, for the SQL database connection, Sql.dll was installed. The following azure database connection string was also deployed in the packages.config file.

```
<connectionStrings>
<add name="connStr" connectionString="Server=tcp:jyfbanksystem.database.windows.net,1433;Initial Catalog=bankdb;Persist Security Info=False;User ID=root1;Password=Jyf980819;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=False;Connection Timeout=30;" />
</connectionStrings>
```

Code snippet 1. Database Connection

4.3 ER Diagram

An entity-relationship diagram (E-R diagram) is a graphical representation of an information system that shows the relationship between people, objects, concepts or events

within that system. An E-R diagram is a data modelling technique that helps define the business process and can be used as the foundation of the relational database. /8/

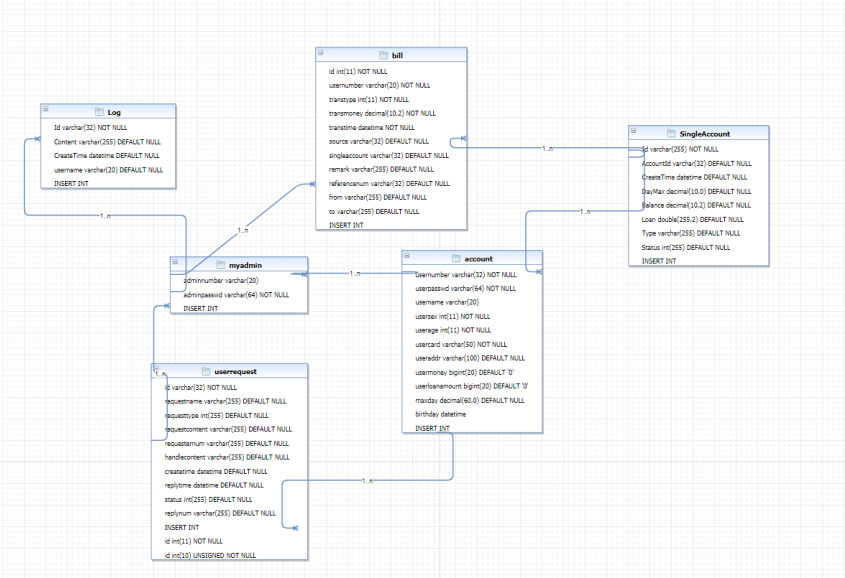


Figure 9. ER diagram

The relationship between each table in the table is described by the ER diagram above.

Tables included in the database:

- Myadmin: This table defines administratornumber and administratorhashed password.
- Account: This table defines all registered bank user personal information. The foreign key referring to the “Accountid” of “SingleAccount” and the id field of “userrequest”.
- SingleAccount: This table defines all bank accounts information in the system. The foreign key referring to the “usernumber” of “bill”.
- Bill: This table shows all transactions information.
- Userrequest: This table users request messages and administrator reply message.
- Log: This table stores delete user’s information and transactions history

4.4 GUI Design

The GUI design was written by CSHTML, CSS, bootstrap.

4.4.1 User Login Page

The user login page is the start point for this application. It has two different pages, one for the bank user and another for the administrator. This page does not require any authentication to be viewed but to proceed to the application, every user needs to log in or register if he/she is a new bank user.

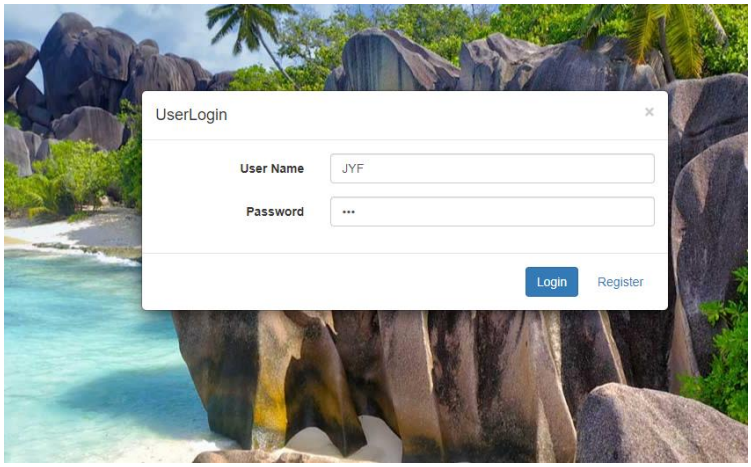


Figure 10. User login page

4.4.2 Administrator Home Page

After administrator logs in successfully, the application will directly go to administrator home page shown below, so that the administrator can proceed with following features:

view bank users' information and account, create new user, deposit to a specific account, view all trading information, approve account register request and reply messages.

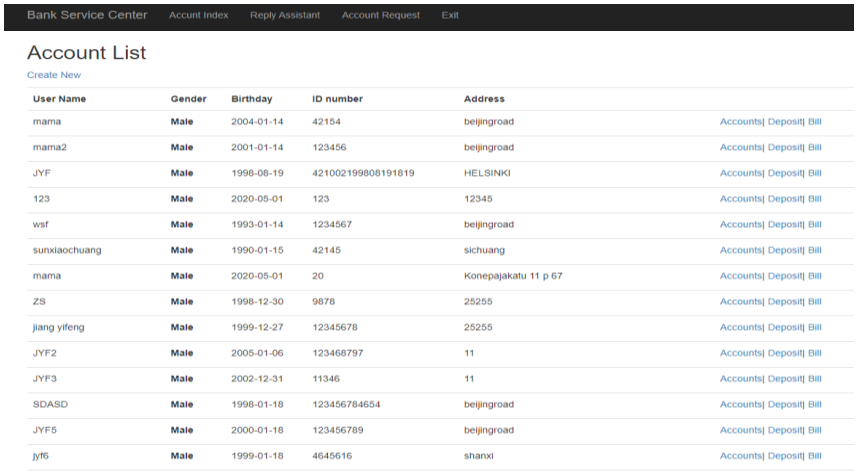


Figure 11. Administrator home page

4.4.3 User Home Page

In this page bank customers can do all the operations for the bank service, such as view/edit personal information/bank account information, view transaction and withdraw and transfer funds to another account.

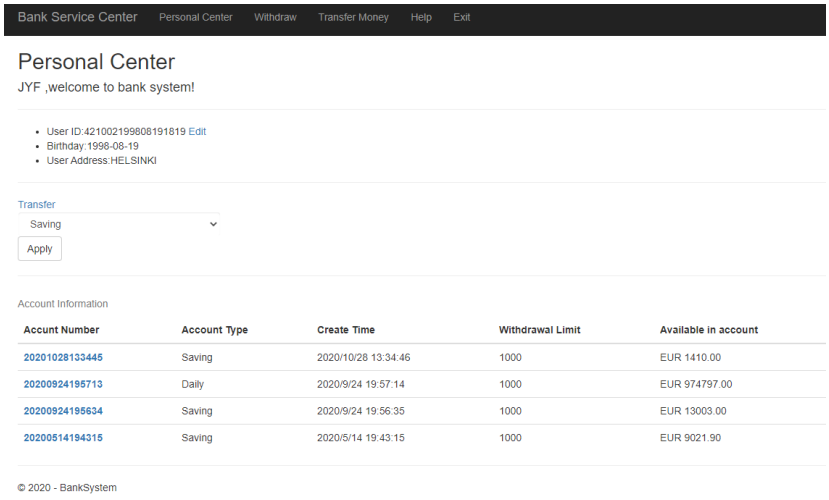


Figure 12. User home page

5 IMPLEMENTATION

5.1 General description of implementation

The implementation section describes the design, management, and analysis of the project in a clear and structured manner. The implementation strategy is included in the implementation plan, business process analysis, scheduling identification, resource or demand identification and integration with the system.

5.1.1 Project structure

The application has been developed and implemented based on the ASP.NET MVC framework. The following picture shows the whole structure for the application

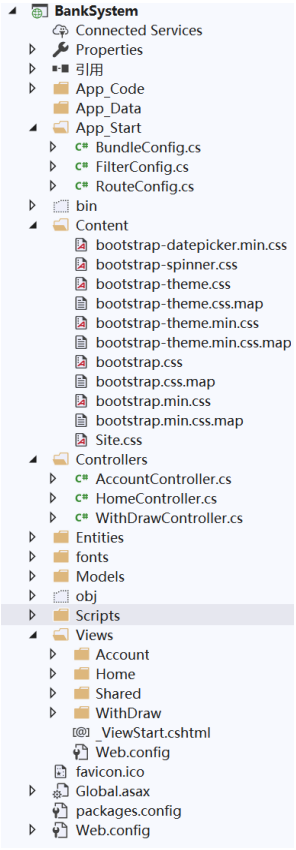


Figure 13. Application Structure

The MVC framework is based on the default nomenclature. The controller is in the controller's folder, the view is in the views folder, and the model is in the model's folder.

Standardized nomenclature reduces the amount of code and helps developers understand the MVC project./9/

Here is a brief overview of the contents of each folder:

- App_Code : App_Code stores all class files that should be compiled dynamically as part of the application. These class files are automatically linked to the application without any explicit instructions or declarations being added to the page to create the dependency. The class file placed in the app code folder can contain any recognizable asp.net components -- custom control, auxiliary class, build provider, business class, custom provider, HTTP handler.
- App_Data: App_Data folder contains the configuration logic files of the application. including bundleconfig.cs, filterconfig.cs, routeconfig.cs.
- Bundleconfig.cs: register the bundled CSS and JS files used in the application.
- Filterconfig. cs: configures external / global filters that can be applied to each Action and Controller.
- Routeconfig.cs: configures the system routing path for MVC applications.
- Content: Stores all static files, such as style sheets (CSS files), icons, and images. The themes folder holds jQuery styles and pictures.
- Controllers: Controllers folder contains all controller classes that handle user input and responses.
- Entities: Entities is to interact with the database for storage purposes.
- Fonts: Fonts folder holds all customized font files.
- Models: Models folder holds all model class files.
- Scripts: Scripts folder store all script JS files supported by the application. By default configuration, Visual Web Developer stores standard MVC, AJAX, and jQuery files in this folder.
- Views: The Views folder used to store HTML files (user interface). It contains corresponding folders for each Controllers folder in this project._Viewstart.cshtml is a start-up file when rendering view files. It will be executed before all views (. cshtml) executed. It is mainly used for some inconvenient or unified operations that cannot be performed in the master (_layout. cshtml).
- Global.asax: it allows writing a code that responds to application level events.

- Packages.config: the packages.config file is managed by NuGet and is used to track the reference packages and package versions installed by the application.
- Web.config: web.config contains database and network configuration.

5.2 Implementation of Functions

The implementation of this application has two main phases: the bank customers' phase and the administrators' phase. All interfaces have been implemented with CSHTML, C#, and a Bootstrap CSS file are used to style and format the front side interface. The database has been carried out with the Azure SQL database.

5.2.1 User Login

The user enters the account and password that are assigned and the QR code authentication. If everything is correct, then it goes to the home page, otherwise an error message will pop-up, the code is given below:

```
 public ActionResult Login(string usernumber,string userpasswd,string authcode)
        {
            if(string.IsNullOrEmpty(usernumber)||   string.IsNullOrEmpty(userpasswd)   ||
string.IsNullOrEmpty(authcode))
            {
                return Json(new { success = false, message = "usernumber,userpassword
authcode is required" });
            }
            HttpCookie cookie = HttpContext.Request.Cookies["tempToken"];
            if(cookie==null)
            {
                return Json(new { success = false, message = "cookies read error" });
            }
            var num= HttpUtility.UrlDecode(cookie.Value, Encoding.GetEncoding("GB2312"));
            if (num!=authcode)
            {
                return Json(new { success = false, message = "auth code expired" });
            }
            using(var db=DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {var account = db.Queryable<Entities.Account>()
                    .Where(u => u.usernumber == usernumber && u.userpasswd == userpasswd)
                    .First();
                if (account==null)
                {
                    return Json(new { success = false, message = "password or user num
error" });}
                Session["usernumber"] = account.usernumber;
                Session["username"] = account.username;
```

```
                return Json(new { success = true, message = "success",data=account });
            }
        }
```

Code snippet 2. User login

5.2.2 QR Code Authentication

In this case, the QR code is the most convenient way to make account safety. To enable this function, the system will first generate a random 4-digital number which makes it a high safety index. Based on the 4-digital number the program will automatically create a QR code for the customer to scan the code.

The QR code function was deployed in a Jquery.qrcode.qr plugin file.

First, the Jquery library file was added and qrcode plugin string to the login file

```
<script type="text/javascript" src="jquery.js"></script>
<script type="text/javascript" src="jquery.qrcode.min.js"></script>
```

Then call the qr plugin by following code:

```
$("document").ready(function () {$.get({
                url: "@Url.Action("AdminAuthCode", "Account")",
                success: function (data) {
                    if (data.success == true) {
                        $('#qrcode').html("");
                        console.log(data.num);
                        jQuery('#qrcode').qrcode({
                            render: "canvas",
                            width: 100,
                            height: 100,
                            text: data.num.toString(),
//Set size of QR Code
                        });
                    }
                }
            });
```

Code snippet 3. QR authentication

On that HTML page a QR code can be directly be generated which allows users to scan by their mobile phone.

5.2.3 SHA-512 Encryption

To make the bank account secure enough, all the passwords of the bank account s will transfer into a SHA-512 hash value and store into the Azure SQL data base. The code for SHA-512 transmission is given below:

```
public class SHAHelper
    {
        public static string CalcSHA(byte[] buffer)
        {
            using (SHA sha = SHA.Create())
            {
                byte[] shaBytes = sha.ComputeHash(buffer);
                return BytesToString(shaBytes);
            }
        }
        private static string BytesToString(byte[] shaBytes)
        {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < shaBytes.Length; i++)
            {
                sb.Append(shaBytes[i].ToString("X2"));
            }
            return sb.ToString();
        }
        public static string CalcSHA(string str)
        {
            //byte[] buffer = Encoding.UTF8.GetBytes(str);
            //return CalcSHA(buffer);
            byte[] bytes = Encoding.UTF8.GetBytes(str);
            byte[] hash = SHA256Managed.Create().ComputeHash(bytes);
            StringBuilder builder = new StringBuilder();
            for (int i = 0; i < hash.Length; i++)
            {
                builder.Append(hash[i].ToString("x2"));
            }
            return builder.ToString();
        }
        public static string CalcSHA(Stream stream)
        {
            using (SHA sha = SHA.Create())
            {
                byte[] buffer = sha.ComputeHash(stream);
                return BytesToString(buffer);
            }
        }
}
```

Code snippet 4. Password encryption

5.2.4 HTTPS Setting

For the security of data transmission for the bank, all the pages and every request send by users should forcedly go through HTTPS layer. To make this function, we need put following code inside the web.config file.

```
protected void Application_BeginRequest(Object sender, EventArgs e)
```

```
{
   if  (HttpContext.Current.Request.IsSecureConnection.Equals(false)  &&  HttpContext.Cur-
rent.Request.IsLocal.Equals(false))
   {
    Response.Redirect("https://" + Request.ServerVariables["HTTP_HOST"]
+   HttpContext.Current.Request.RawUrl);
   }
}
```

Code snippet 5. HTTPS setting

5.2.5 User Registration

Both the user and the administrator can create a new bank account. The new customer needs to create a new bank account to be able to use the bank service. After successful registration, the data of the user will be stored in the database. A function "Create" will be responsible for the processing. The new users/customers have to fill all the required information into the form and the age and ID number must be checked, then it is possible to register in. The code is given below:

```
public ActionResult Create(FormCollection collection)
 {
try
            {
                using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
                {
var account = new Entities.Account
                    { useraddr = collection["address"],
                        birthday = DateTime.Parse(collection["birthday"]),
                        usercard = collection["idcard"],
                        userloanamount = 0,
                        usermoney = 0,
                        username = collection["username"],
                        usernumber = DateTime.Now.ToString("yyyyMMddHHmmss"),
                        userpasswd = SHAHelper.CalcSHA(collection["userpasswd"]),
                        usersex = int.Parse(collection["usersex"]),
                        maxday = 1000
                    };
                    int row = db.Insertable(account).ExecuteCommand();
                    var sc = new Entities.SingleAccount
                    {AccountId = account.usernumber,
                        Balance = 1000,
                        CreateTime = DateTime.Now,
                        DayMax = 1000,
                        Id = DateTime.Now.ToString("yyyyMMddHHmmss"),
                        Loan = 0,
                         Type= collection["type"],
                          Status=0
```

```
                };
                row = db.Insertable(sc).ExecuteCommand();
                if (row > 0)
                {
                    return Json(new { success = true, message = "add success" });
                }
                return Json(new { success = false, message = "add falied" });
            }
        }
        catch (Exception ex)
        {
            return Json(new { success = false, message = ex.Message });
        }
    }
```

Code snippet 6. Create a new user

5.2.6 Edit User Information

The bank users can edit their address and change their password and withdrawal limitation, A function called “Edit” will handle all the process. The code is given below

```
  public ActionResult Edit( FormCollection collection)
        {
            try
            {
                using(var db=DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
                {
                    var useraddr = collection["address"];
                    var username = collection["username"];
                    var usersex = int.Parse(collection["usersex"]);
                    var usercard = collection["idcard"];
                    var userpasswd = SHAHelper.CalcSHA(collection["userpasswd"]);
                    var userage = int.Parse(collection["userage"]);
                    var maxday = int.Parse(collection["maxday"]);
                    var row = db.Updateable<Entities.Account>().SetColumns(u => new Enti-
ties.Account
                    {
                        useraddr = useraddr,
                        username = username,
                        usersex = usersex,
                        usercard = usercard,
                        userpasswd = userpasswd,
                        userage = userage,
                         maxday= maxday
                    }).Where(u => u.usernumber == collection["usernumber"]).ExecuteCom-
mand(); ;
                    if (row > 0)
                    {
                        return Json(new { success = true, msg = "Update successfully" });
                    }
                    else
```

```
                { return Json(new { success = false, msg = "Update failed" }); }
            }

            return RedirectToAction("Index");
        }
        catch(Exception ex)
        {
            return Json(new { success = false, msg =ex.Message });
        }
    }
```

Code snippet 7. Edit user information

5.2.7 Delete User Function

The administrator has right to delete the account. A function called “DeleteSingleAccount” will handle this process. The code for this method is given below.

```
public ActionResult DeleteSingleAccount(string id)
    {
        try
        {
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                int row = db.Deleteable<Entities.SingleAccount>()
                    .Where(u => u.Id == id)
                    .ExecuteCommand();

                if(row>0)
                {
                    return Json(new { success = true, message = "success" });
                }
                else
                {
                    return Json(new { success = false, message = "failed" });
                }
            }
        }
        catch
        {
            return View();
        }
    }
```

Code snippet 8. Delete user account

5.2.8 Transfer Funds

Transfer money function enables users to transfer money to a specific account stored in the database. A method called “TransMoney” will handle all the process. Users should

make input the target account, target username and password to make transactions. If the user target username and target account can match a specific account in the database and there is enough money, then the transactions will be accepted. Otherwise, the transaction will not be handled and an error message dialog will pop-up, the code for this function is given below:

```
public ActionResult TransMoney(string accounts, string targetNum, string targetName, string
password, decimal transferMoney, string message, string referenceNum)
        {
            var number = Session["usernumber"].ToString();
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                var singleAccount = db.Queryable<Entities.SingleAccount>()
                    .Where(u => u.Id == accounts && u.Status == 1)
                    .First();
                if (singleAccount.Balance < transferMoney)
                {
                    return Json(new { success = false, message = "balance not enough" });
                }
                var getDayWithDrawMoney = db.Queryable<Bill>()
                  .Where(u => SqlSugar.SqlFunc.DateIsSame(u.transtime, DateTime.Now))
                  .Where(u => u.singleaccount == accounts)
                  .Where(u => u.transtype == 1 || u.transtype == 3)
                  .Sum(u => u.transmoney);
                if (getDayWithDrawMoney >= singleAccount.DayMax)
                {
                    return Json(new { success = false, message = "max money have handled" });
                }
                var targetSingleAccount = db.Queryable<Entities.SingleAccount>()
                    .Where(u => u.Id == targetNum && u.Status == 1)
                    .First();
                if (targetSingleAccount == null)
                {
                    return Json(new { success = false, message = "target account not ex-
ist" });
                }
                if (singleAccount == null)
                {
                    return Json(new { success = false, message = "single account not ex-
ist" });
                }
                var account = db.Queryable<Account>().Where(u => u.usernumber == singleAc-
count.AccountId).First();
                if (account.userpasswd != SHAHelper.CalcSHA(password))
                {
                    return Json(new { success = false, message = "password error" });
                }
                var targetAccount = db.Queryable<Account>()
                 .Where(u => u.usernumber == targetSingleAccount.AccountId)
                 .First();
```

```
            if (targetAccount.username != targetName)
                {
                    return Json(new { success = false, message = "target name error" });
                }
                if (singleAccount.Balance < transferMoney)
                {
                    return Json(new { success = false, message = "money error" });
                }
                singleAccount.Balance = singleAccount.Balance - transferMoney;
                targetSingleAccount.Balance = targetSingleAccount.Balance + transferMoney;
                Bill output = new Bill
                {
                    transmoney = transferMoney,
                    transtime = DateTime.Now,
                    transtype = 3,
                    usernumber = targetAccount.usernumber,
                    singleaccount = targetSingleAccount.Id,
                    referencenum = referenceNum,
                    source = singleAccount.Id,
                    remark = message,
                    From = singleAccount.AccountId,
                    To = targetSingleAccount.AccountId
                };
                Bill source = new Bill
                {
                    transmoney = -transferMoney,
                    transtime = DateTime.Now,
                    transtype = 4,
                    usernumber = singleAccount.AccountId,
                    singleaccount = singleAccount.Id,
                    source = targetSingleAccount.Id,
                    referencenum = referenceNum,
                    remark = message,
                    From = singleAccount.AccountId,
                    To = targetSingleAccount.AccountId
              };
                try
                {
                    db.BeginTran();
                    db.Insertable<Bill>(output).ExecuteCommand();
                    db.Insertable<Bill>(source).ExecuteCommand();
                    db.Updateable<Entities.SingleAccount>()
                       .SetColumns(u => new SingleAccount { Balance = singleAccount.Bal-
ance })
                       .Where(u => u.Id == singleAccount.Id)
                       .ExecuteCommand();
                    db.Updateable<SingleAccount>()
                     .SetColumns(u => new SingleAccount { Balance = targetSingleAc-
count.Balance })
                     .Where(u => u.Id == targetSingleAccount.Id)
                     .ExecuteCommand();
                    db.CommitTran();
                }
```

```
            catch (Exception ex)
            {
                db.RollbackTran();
                return Json(new { success = false, message = "database error" });
            }
            return Json(new { success = true, message = "transfer success" });
        }
    }
```

Code snippet 9. Transfer funds

5.2.9 Administrator Deposit

Only the administrator has the right to deposit to a specific account. A method called "Deposit" will handle this process. The code is shown below

```
public ActionResult Deposit(string account, decimal Money)
    {

        using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
        {


            var sc = db.Queryable<Entities.SingleAccount>()
                .Where(u => u.Id == account).First();

            if (sc == null)
            {
                return Json(new { success = false, message = "account error" });
            }


            sc.Balance = sc.Balance + Money;


            var row = db.Updateable<Entities.SingleAccount>()
                .SetColumns(u => new Entities.SingleAccount { Balance = sc.Balance })
                .Where(u => u.Id == account)
                .ExecuteCommand();

            var bill = new Bill();
            bill.transmoney = Money;
            bill.transtime = DateTime.Now;
            bill.transtype = 1;
            bill.usernumber = sc.AccountId;
            bill.singleaccount = account;
            bill.source = "AdministratorDeposit";
            bill.From = string.Empty;
            bill.To = account;
```

```
                bill.remark = $"Deposit Money:{Money}";
                bill.referencenum = DateTime.Now.ToString("yyyyMMddHHmmssfff");
                int bbb = db.Insertable(bill).ExecuteCommand();

                return Json(new { success = true, message = " deposit success", UserNumber
= sc.AccountId });

            }

        }
```

Code snippet 10. Administrator deposit

5.2.10 Withdrawal

A bank user has the right to withdraw money from a bank account. First, select a bank account is selected and the amount of money is put in then the server will check whether there is enough money or not and if the amount is less than the withdrawal limit. If everything is positively correct, then withdraw will process successfully. Otherwise, an error message will be shown.

```
public ActionResult WithDraw(string accounts, decimal Money)
        {
            var number = Session["usercard"].ToString();
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
            var sc = db.Queryable<Entities.SingleAccount>()
                    .Where(u => u.Id == accounts).First();
                if (sc == null)
                {
                    return Json(new { success = false, message = "account error" });
                }
                if (sc.Balance < Money)
                {
                    return Json(new { success = false, message = "mony is not enough" });
                }
                if (sc.DayMax < Money)
                {
                    return Json(new { success = false, message = "max money have handled" });
                }
                var getDayWithDrawMoney = db.Queryable<Bill>()
                    .Where(u => SqlSugar.SqlFunc.DateIsSame(u.transtime, DateTime.Now))
                    .Where(u => u.usernumber == sc.Id)
                    .Where(u => u.transtype == 1 || u.transtype == 3)
                    .Sum(u => u.transmoney);
                if (getDayWithDrawMoney >= sc.DayMax)
                {
                    return Json(new { success = false, message = "max money have handled" });
                }
                sc.Balance = sc.Balance - Money;
```

```
                var row = db.Updateable<Entities.SingleAccount>()
                    .SetColumns(u => new Entities.SingleAccount { Balance = sc.Balance })
                    .Where(u => u.Id == accounts)
                    .ExecuteCommand();
                var bill = new Bill();
                bill.transmoney = -Money;
                bill.transtime = DateTime.Now;
                bill.transtype = 0;
                bill.usernumber = sc.AccountId;
                bill.singleaccount = accounts;
                bill.source = "With Draw";
                bill.From = sc.AccountId;
                bill.To = string.Empty;
                bill.remark = $"With Draw Money:{Money}";
                bill.referencenum = DateTime.Now.ToString("yyyyMMddHHmmssffff");

                int bbb = db.Insertable(bill).ExecuteCommand();

                return Json(new { success = true, message = "withdrawsuccess", singleac-
count = sc.Id });

            }

        }
```

Code snippet 11. Withdrawal function

5.2.11 Transaction Details Function

In the transaction details module. Users can view all transaction details here included the trading amount, trading time, and trading source. The code is shown below.

```
public ActionResult WithDrawBillDetail(int id)
        {
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                var bill = db.Queryable<Entities.Bill>()
                    .Where(u => u.id == id)
                    .First();

                var accountFrom = db.Queryable<SingleAccount, Account>((sa, a) => new ob-
ject[]{
        JoinType.Left,sa.AccountId==a.usernumber}).Where((sa,   a)   =>   a.usernumber   ==
bill.From)
        .Select((sa, a) => new { Name = a.username }).First();

                if (accountFrom == null)
                {
                    ViewBag.From = "NULL";
                }
                else
                {
```

```
                ViewBag.From = accountFrom.Name;
            }

            var accountTo = db.Queryable<SingleAccount, Account>((sa, a) => new ob-
ject[]{
        JoinType.Left,sa.AccountId==a.usernumber}).Where((sa,  a)  =>  a.usernumber  ==
bill.To)
        .Select((sa, a) => new { Name = a.username }).First();

            if (accountTo == null)
            {
                ViewBag.To = "NULL";
            }
            else
            {
                ViewBag.To = accountTo.Name;
            }
            return View(bill);
        }
```

Code snippet 12. Transaction detail page

5.2.12 Search Trading Record

A method called “WithDrawBillQuery” will handle the search process. If the bank user chooses the start date/end date and trading type, then the result will display to the bank user.

```
public ActionResult WithDrawBillQuery(DateTime start, DateTime end, int type, string
singleaccount)
        {

            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                var singleAccount = db.Queryable<SingleAccount>()
          .Where(u => u.Id == singleaccount)
          .First();

                var bills = db.Queryable<Entities.Bill>()

                    .Where(u => u.singleaccount == singleaccount)
                    .Where(u => u.transtime >= start && u.transtime <= end)
                    .WhereIF(type != 5, u => u.transtype == type)
                    .ToList();

                var user = db.Queryable<Entities.Account>()
                    .Where(u => u.usernumber == singleAccount.AccountId)
                    .First();
                ViewBag.SingleAccount = singleAccount;
                ViewBag.Account = user;
                if (type == 5)
```

```
                {
                }
                ViewBag.Type = type;
                return View("WithDrawBill", bills);
            }
        }
```

Code snippet 13. Search function

5.2.13 Message Function

The message function is deployed in both the Controllers part and Views part. The controller part below handles the message communication processes.

```
public class UserRequestController : Controller
    {
        // GET: UserRequest
        public ActionResult UserIndex()
        {
            var number = Session["usernumber"].ToString();
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                var requests = db.Queryable<UserRequest>()
                    .Where(u => u.requesternum == number)
                    .ToList();
                return View(requests);
            }
        }
        public ActionResult AdminIndex()
        {
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                var requests = db.Queryable<UserRequest>()
                    .ToList();
                return View(requests);
            }
        }
        public ActionResult CreateUserRquest()
        {
            return View();
        }
        [HttpPost]
        public ActionResult CreateUserRquest(string requestname,string requestcontent)
        {
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                var number = Session["usernumber"].ToString();
                var request = new UserRequest
                {
                    createtime = DateTime.Now,
                    id = Guid.NewGuid().ToString("N"),
                    requestcontent = requestcontent,
```

```
                    requestname = requestname,
                    requesternum = number,
                    status=0
                };
                if (db.Insertable(request).ExecuteCommand() > 0)
                {
                }
                return RedirectToAction("UserIndex");
            }
        }
        public ActionResult CeateRepy(string id)
        {
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                var requests = db.Queryable<UserRequest>()
                    .Where(u => u.id == id)
                    .First();
                return View(requests);
            }
        }
        [HttpPost]
        public ActionResult CeateRepy(string id,string requestname, string requestcontent,
string requesternum,string handlecontent)
        {
            string adminNum = Session["AdminUsernumber"].ToString();
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                var requests = db.Updateable<UserRequest>()
                    .SetColumns(u => new UserRequest { handlecontent = handlecontent, sta-
tus = 1, replynum = adminNum,replytime=DateTime.Now })
                    .Where(u => u.id == id).ExecuteCommand();
                   return RedirectToAction("AdminIndex");
            }
        }
        public ActionResult DeleteRepy(string id)
        {
            return View();
        }
        public ActionResult Details(string id)
        {
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                var requests = db.Queryable<UserRequest>()
                    .Where(u => u.id == id)
                    .First();
                return View(requests);
            }
        }
```

```
}
```

Code snippet 14. Message function

In addition, for users to see the messages records, the views part can show the message information such as request time, reply time, message status (to get the reply/send a reply or not), and the content of the message.

```
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.requestname)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.requestcontent)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.handlecontent)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.createtime)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.replytime)
        </th>
    </tr>
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.requestname)
        </td>

        <td>
            @Html.DisplayFor(modelItem => item.requestcontent)
        </td>

        <td>
            @Html.DisplayFor(modelItem => item.handlecontent)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.createtime)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.replytime)
        </td>
    </tr>
}
```

```
</table>
```

Code snippet 15. Message detail page

5.2.14 Applying for a New Account Function

The code below in the application controller class describes applying for a new account in the system. The method called “CreateUserRequest” will send the request to the administrator and the “Approve” and “GetUnApproveAccount” will allow the administrator to handle the account requests, after administrator approves the new account will be created successfully and displayed on the user’s homepage. The code is shown below.

```
public ActionResult AppNewAccount(string usernumber, string type)
        {
            var sc = new Entities.SingleAccount
            {
                AccountId = usernumber,
                Balance = 1000,
                CreateTime = DateTime.Now,
                DayMax = 1000,
                Id = DateTime.Now.ToString("yyyyMMddHHmmss"),
                Loan = 0,
                Type = type,
                Status = 0,
            };
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                int row = db.Insertable(sc).ExecuteCommand();
            }
            return Json(new { success = false, message = "apply success" });
        }
        public ActionResult Approve(string id)
        {
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                int row = db.Updateable<Entities.SingleAccount>()
                    .SetColumns(u => new SingleAccount { Status = 1 })
                    .Where(u => u.Id == id)
                    .ExecuteCommand();
                if (row > 0)
                {
                    return Json(new { success = true, message = "approve success" });
                }
                return Json(new { success = false, message = "approve failed" });
            }
        }
        public ActionResult GetUnApprovedAccount()
        {
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
```

```
            var singleAccounts = db.Queryable<SingleAccount>()
                   .Where(u => u.Status == 0)
                   .ToList();
            return View(singleAccounts);
        }
    }
```

Code snippet 16. Applying for a new account

5.2.15 Block Account Function

The administrators are able to block a bank account. After blocking, no transaction can be made on this the bank account until administrators approve it. To enable this function, the code is shown below.

```
public ActionResult BlockAccount(string id)
    {
        try
        {
            using (var db = DbFactory.GetSqlSugarClient(WebConfig.ConnStr))
            {
                int row = db.Updateable<Entities.SingleAccount>()
                    .SetColumns(it=>new SingleAccount { Status=0})

                     .Where(u => u.Id == id)
                     .ExecuteCommand();
                if (row > 0)
                {
                    return Json(new { success = true, message = "Success" });
                }
                else
                {
                    return Json(new { success = false, message = "Failed" });
                }
            }
        }
        catch
        {
            return View();
        }
    }
```

Code snippet 17. Block account

6 APPLICATION TESTING

In order to make sure the quality and find any errors in the application. Software testing was carried out.

Software testing is used to confirm whether the performance or quality of a program meets some requirements that are put forward before development.

The application was tested with Windows 10 platform by using Visual Studio 2019.

This chapter describes the test cases for both administrator and user modules.

6.1 User and Administrator Login Page

There are two different login pages, one for the bank customers, another one is for the administrator. Both pages share the same design. When the users enter a username and password, the system will determine the password. If the password matches, the QR code dialog will pop-up and do the double security check.

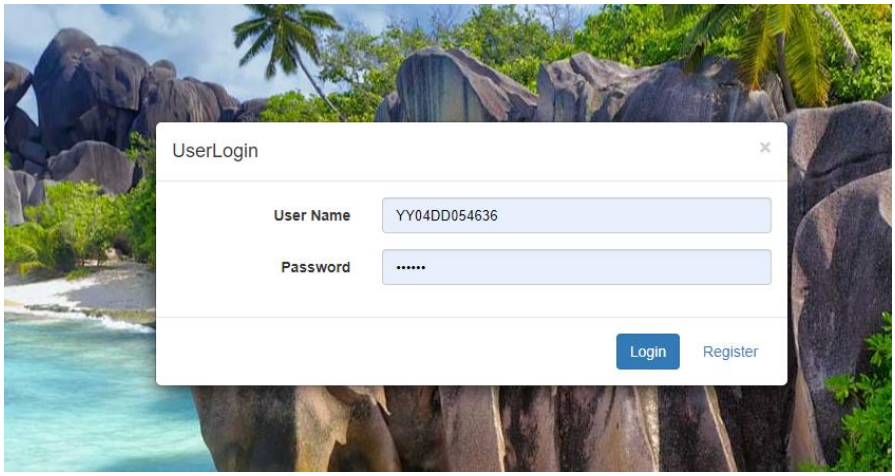


Figure 14. Login page

The application will generate a QR code on every log-in attempt land a mobile device can be used to scan the QR code to get the security number. The figure below shows a mobile screenshot that successfully gets the security number by scanning QR code.

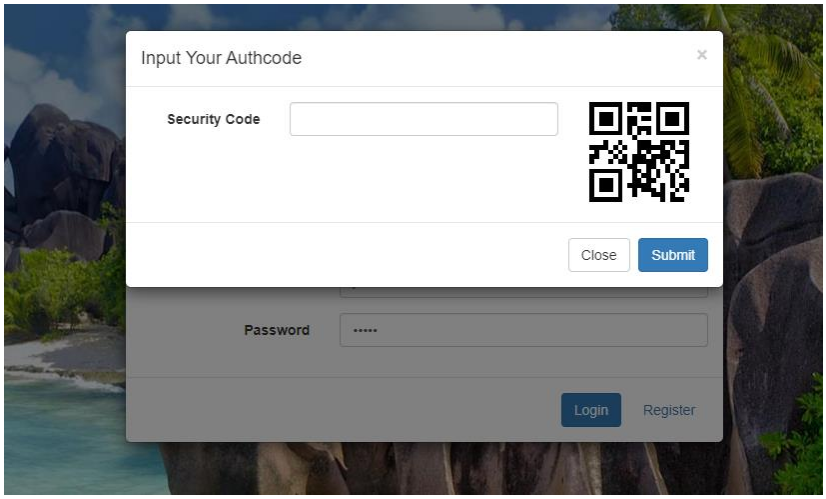


Figure 15. QR code

Figure 16. Scanning result

6.2 Bank Account Register Page

In the set-up user page, a new account can be registered once you the personal information has been entered and the “Create” button pressed. The system will generate a bank account to the user which allows the log into the bank service center. The figure below shows the result of the account generation process.

To create an account, the user must be over 18 and the id number must be unique.

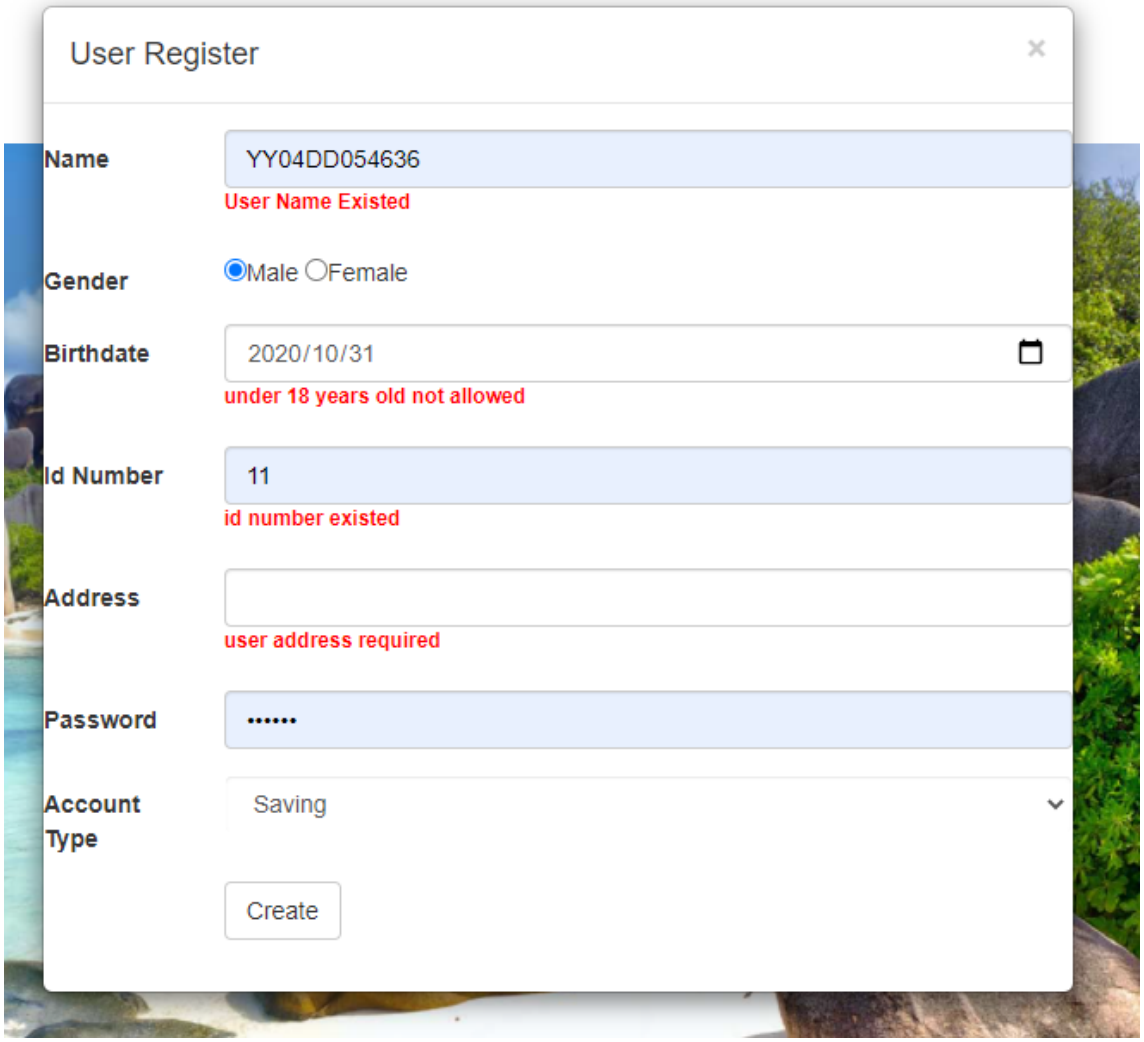


Figure 17. Register page

6.3 Edit User Information Page

On the home page for bank users, all account information is displayed there, and there is an edit button to click.

The figure below shows the user home page with account information.

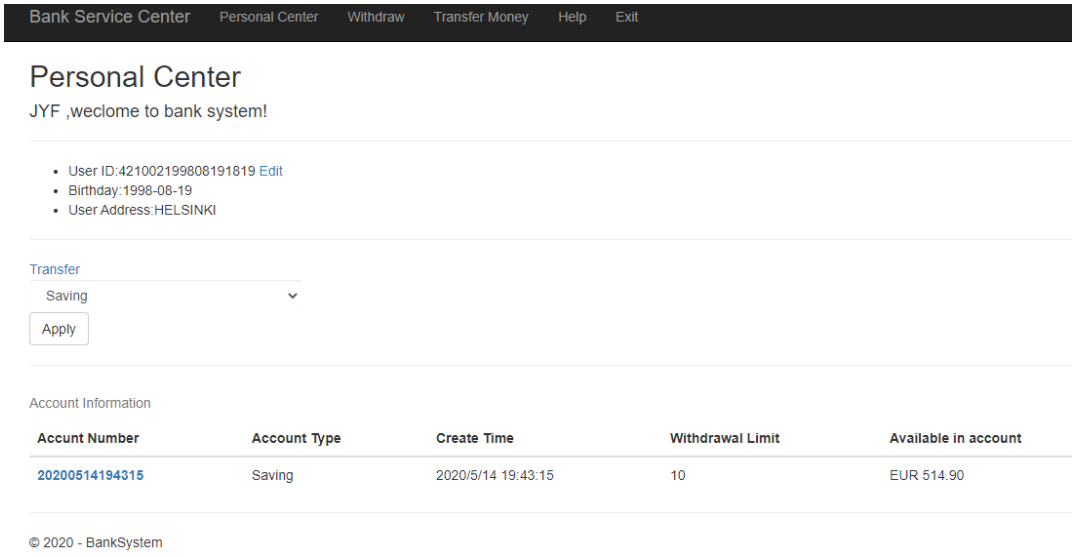


Figure 18. Bank user home page

The edit page allows bank users to edit their home address and the withdrawal limit. The figure below shows the edit page.

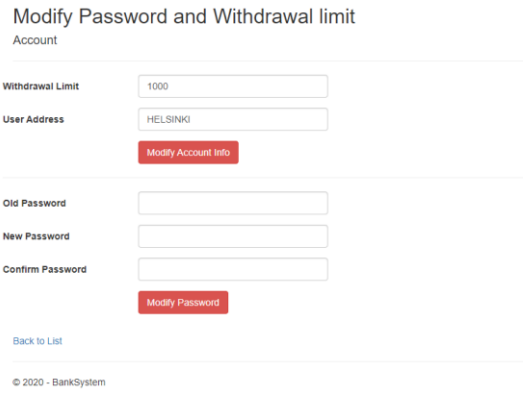


Figure 19. Edit page

The following figure shows the pop-up dialog box after the edit completed.

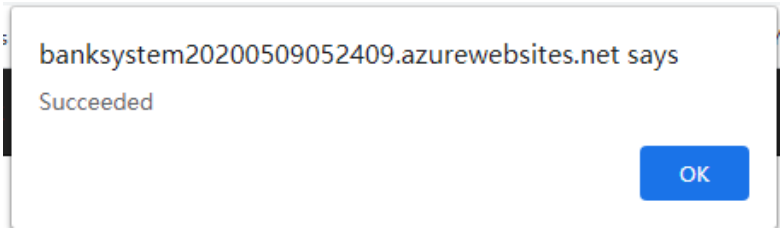


Figure 20. Edit dialog box

6.4 Withdrawal Page

The figure below shows the withdrawal page for the bank user. The account to withdraw money from must be chosen first and the amount of money entered.

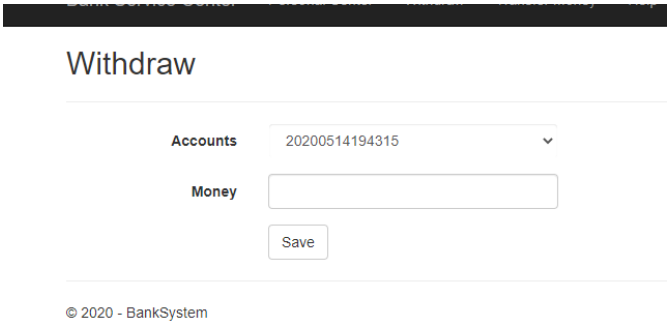


Figure 21. Withdrawal page

The figure below shows that withdrawal failed because the trading amount exceed the trading limitation.

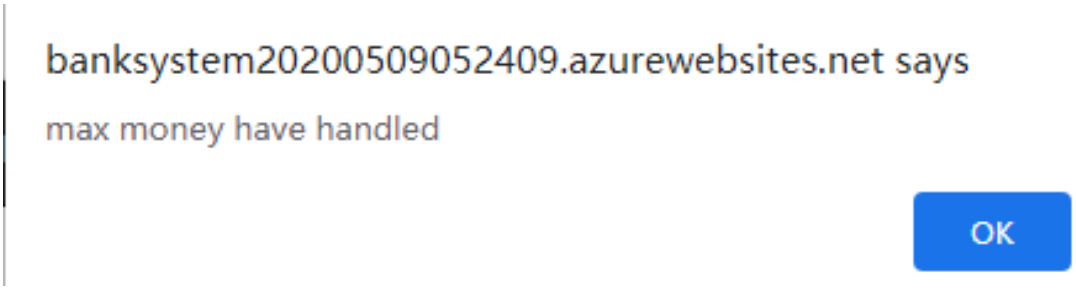


Figure 22. Withdrawal dialog

6.4.1 Deposit Page

The figure below is the Deposit page for the administrator. First the account to deposit m oney into must be chosen and the amount of money entered. After clicking the “save” button, the system will deposit money into the target account , and then a “Deposit Succ ess” message will pop-up.

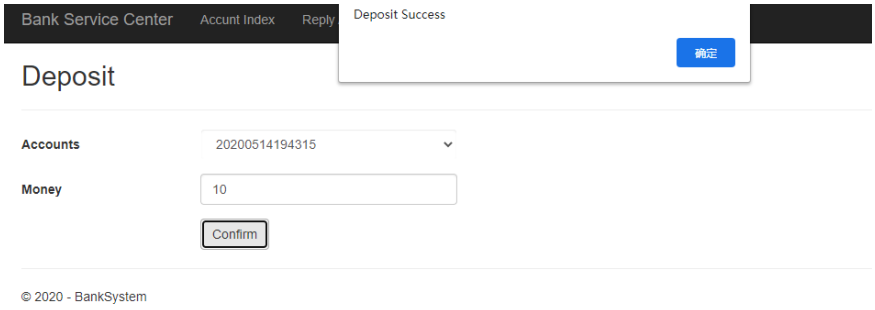


Figure 23. Deposit Success

6.5 Transactions Page

On this page, bank users can see their transaction (deposit, withdraw and transfer funds) inquiry by time. First, there is a dialog to choose start-time and end-time, and then the trading type is chosen. By clicking the search button, the filter will apply. The figure shows the calendar component.

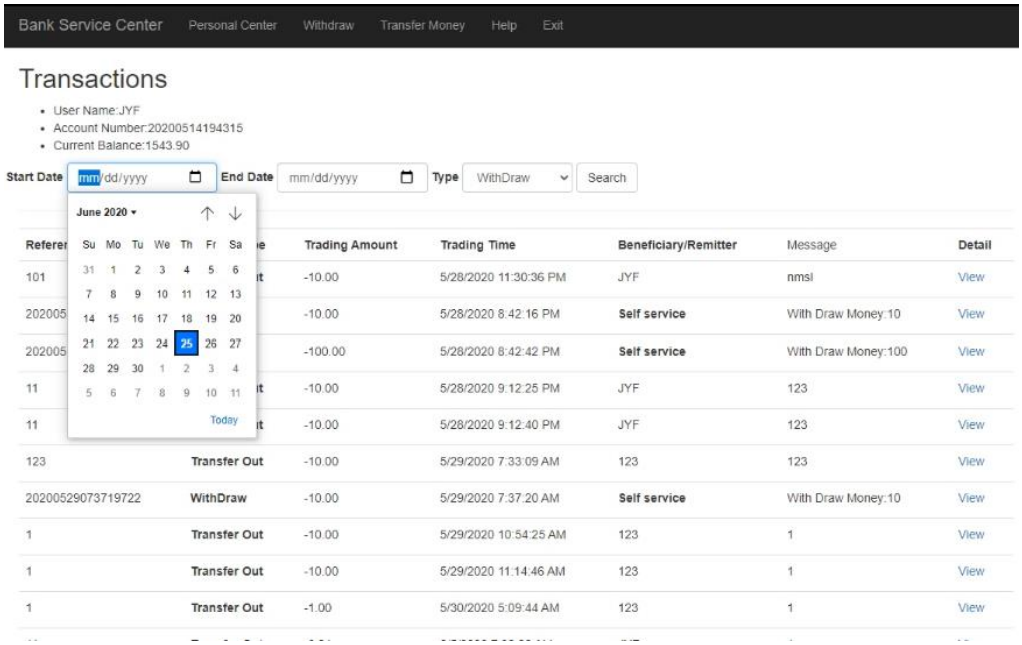


Figure 24. Choose a time period and trading type

The following table shows the search results.

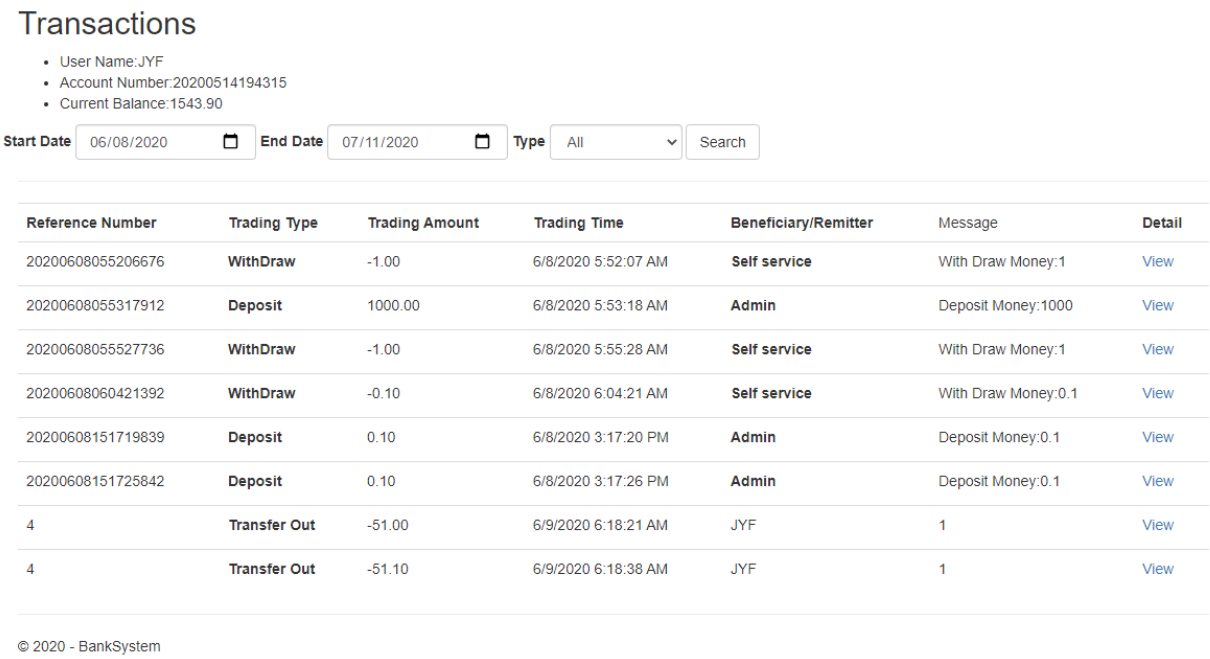


Figure 25. Inquiry page

6.5.1 Transaction Detail Page

On the transactions page, in every trading record, there is a “view” link. Clicking the “view will redirect to a transactions detail page which displays all transactions detail. The transaction detail page picture is shown below.

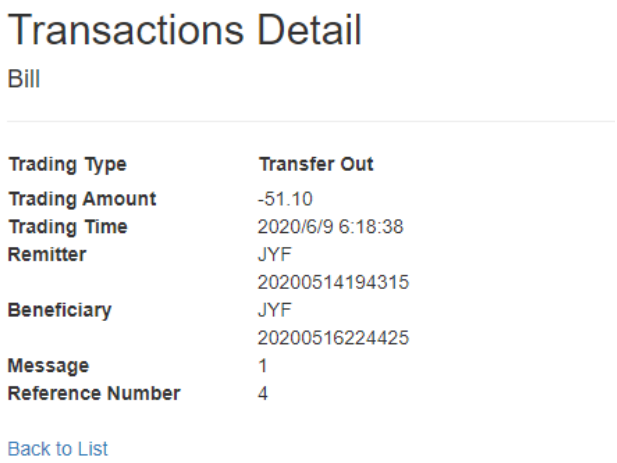


Figure 26. Transaction detail page

6.6 Transfer Money page

The transfer money web page is shown below. To transfer money, the correct target account must be entered and then the target name will pop-up, as well as the password to make sure the safety. Finally, the confirm button is clicked. After the verification, the money will transfer successfully.

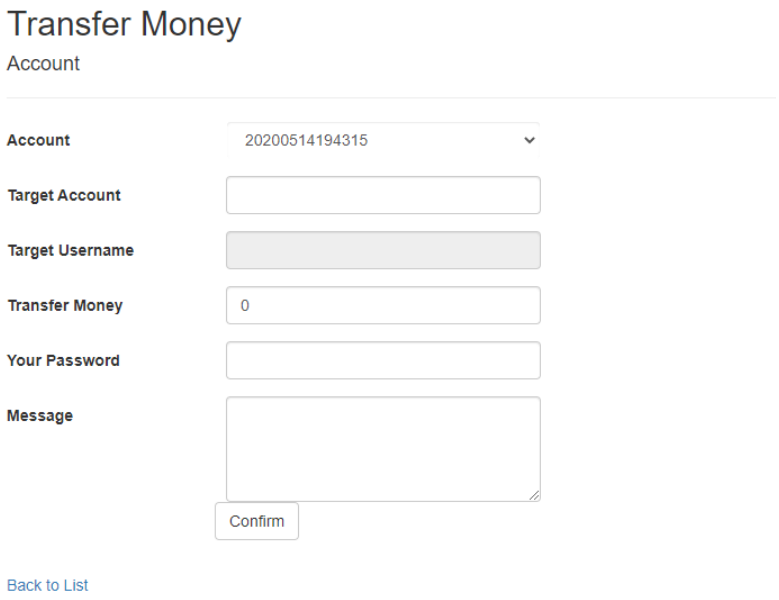


Figure 27. Transfer money page

6.7 Message Page

On the message page the bank users can send request information to the administrators such as delete my account. In addition, the administrator can reply the user. Figures 28 and 29 below shows how to make the request/reply.

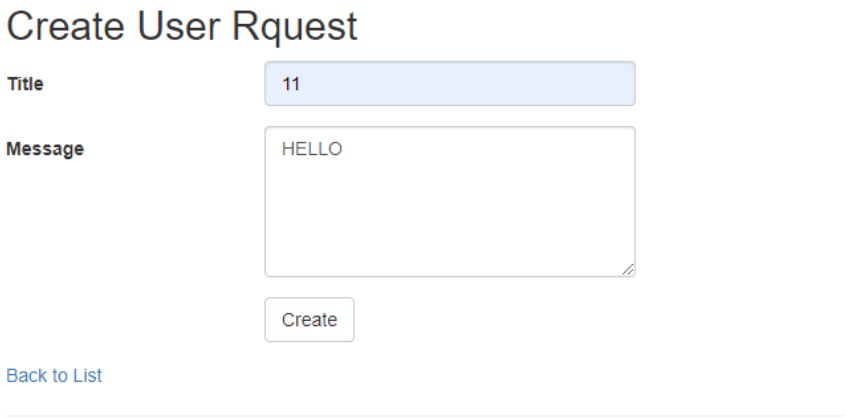


Figure 28. Create user request

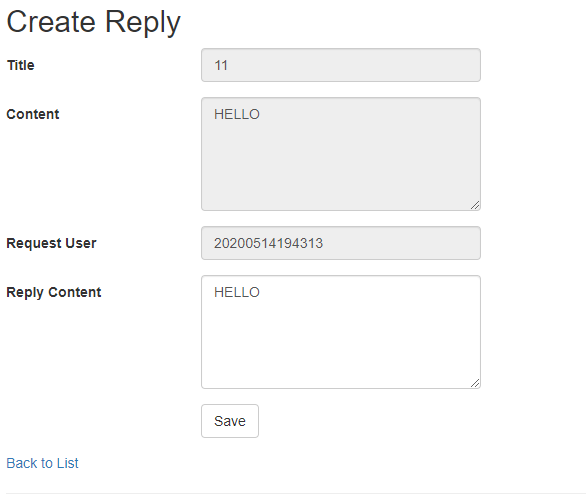


Figure 29. Reply to a request

The message page also enables the viewing of the message records. The following figure shows the message records for an administrator.

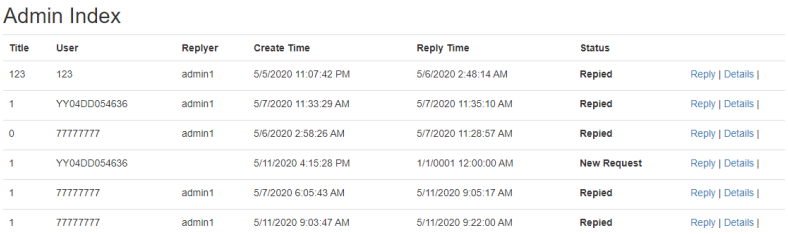
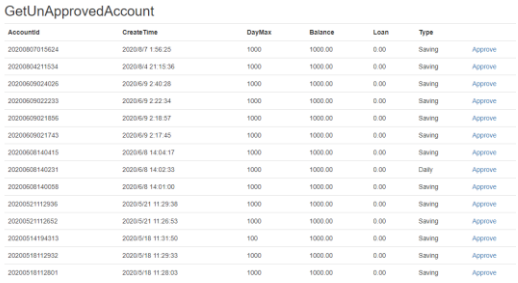


Figure 30. Message records

6.7.1 Apply for a New Account Module

On the home page, users can choose to apply for a daily account or a saving account. Then the request will send to the administrators, on the Account Request page, there is an “Apply Account” button, the status of the account will go to 1 and disappear from the list. Then the account will be stored into the database and a new account has been successfully created.


GetUnApprovedAccount

AccountId	CreateTime	DayMax	Balance	Loan	Type	
20200807015624	2020/8/7 1:56:25	1000	1000.00	0.00	Saving	Approve
20200804211534	2020/8/4 21:15:36	1000	1000.00	0.00	Saving	Approve
20200609024026	2020/6/9 2:40:28	1000	1000.00	0.00	Saving	Approve
20200609022233	2020/6/9 2:22:34	1000	1000.00	0.00	Saving	Approve
20200609021856	2020/6/9 2:18:57	1000	1000.00	0.00	Saving	Approve
20200609021743	2020/6/9 2:17:45	1000	1000.00	0.00	Saving	Approve
20200608140415	2020/6/8 14:04:17	1000	1000.00	0.00	Saving	Approve
20200608140231	2020/6/8 14:02:33	1000	1000.00	0.00	Daily	Approve
20200608140058	2020/6/8 14:01:00	1000	1000.00	0.00	Saving	Approve
20200521112936	2020/5/21 11:29:38	1000	1000.00	0.00	Saving	Approve
20200521112652	2020/5/21 11:26:53	1000	1000.00	0.00	Saving	Approve
20200514194313	2020/5/18 11:31:50	100	1000.00	0.00	Saving	Approve
20200518112932	2020/5/18 11:29:33	1000	1000.00	0.00	Saving	Approve
20200518112801	2020/5/18 11:28:03	1000	1000.00	0.00	Saving	Approve

Figure 31. Account request page

The dialog comes out after administrator click approve button.

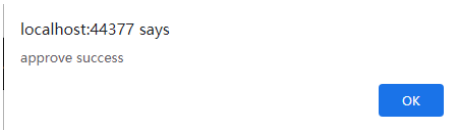


Figure 32. Approve success dialog

7 CONCLUSION

This thesis project aimed to develop an application, which simulates an efficient and convenient bank management system, to help both the bank customer and bank administrator to manage their accounts more use-friendly and allows them to do the bank-related business. This application provides various features. Bank customers can make virtual bank transactions, view bank transactions, manage their accounts and send messages to the administrators. For the bank administrators, this application allows administrators to manage bank customers' account, view bank history, add virtual money to a certain account and reply bank customers' messages. To achieve this goal, this application was based on.NET framework, using C# and Azure SQL database. On the other hand, to keep the safety of the bank management, web-pages are using HTTPS and password is encrypted by SHA-512.

During the development process, the most challenging part is to ensure the safety of the bank system in an effective way. To achieve this requirement, QR code was used as the second authentication method other than encrypted. Since QR code technology was a new technology for me, it brought some difficulties. Embedding a QR code into the login system was time-consuming as was finding a way to make it more convenient for the bank users.

In conclusions, this application was developed in strict accordance with the project specification. After demand analysis, brief design, detailed design, coding and testing, the scheduled business functions were finally completed as scheduled.

7.1 Future Work

However, the current bank user management system obviously has many shortcomings, for example the UI interface design is not elegant enough. In addition, some functions could have been done better. For example, the password only was encrypted to SHA-512, but it would be better to encrypt messages as well. Some new functions could be added as well, such as loans management. At the same time, if the software is to be put into practical use, it should be considered how to operate and maintain it, how to manage its users, and how to improve the efficiency of data processing. This needs to think from a deeper

business perspective to make the application of this management system more perfect, intelligent, practical.

REFERENCES

/1/ C #. Wikipedia. Accessed 24.4.2020. https://en.wikipedia.org/wiki/C_Sharp_(programming_language)

/2/ What is ASP.NET? Microsoft. Accessed 4.5.2020. https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet

/3/ Get Bootstrap. Accessed 1.6.2020. https://getbootstrap.jp/

/4/ Azure SQL database. Microsoft. Accessed 11.9.2020. https://azure.microsoft.com/en-us/services/sql-database/

/5/ B-S structure model. Research gate. Accessed 1.5.2020. https://www.researchgate.net/figure/Structure-of-3-tier-B-S-model-in-the-data-management-module_fig3_220828005

/6/ SHA-512. Wikipedia. Accessed 1.10.2020. https://en.bitcoinwiki.org/wiki/SHA-512

/7/ UML. Geeks for geeks. Accessed 8.4.2020 https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams

/8/ ERD. TechTarget. Accessed 14.4.2020 https://searchdatamanagement.techtarget.com/definition/entity-relationship-diagram-ERD

/9/ ASP.NET MVC. CSDN. Accessed 24.4.2020 https://blog.csdn.net/nanbaifeiliao/article/details/83339324