

Project documentation and implementation artifacts

Mojo Banking Solutions, Inc.



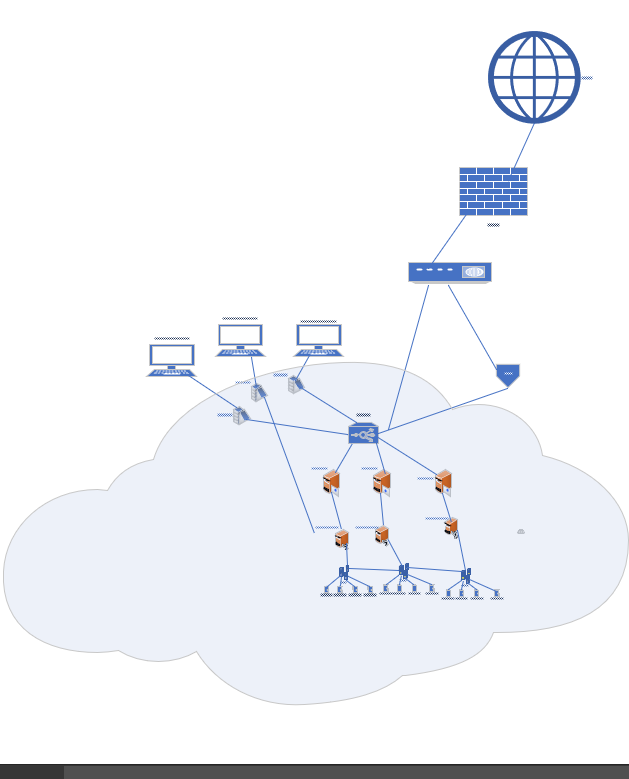
October 12, 2020

Shane Ẽire Byrne

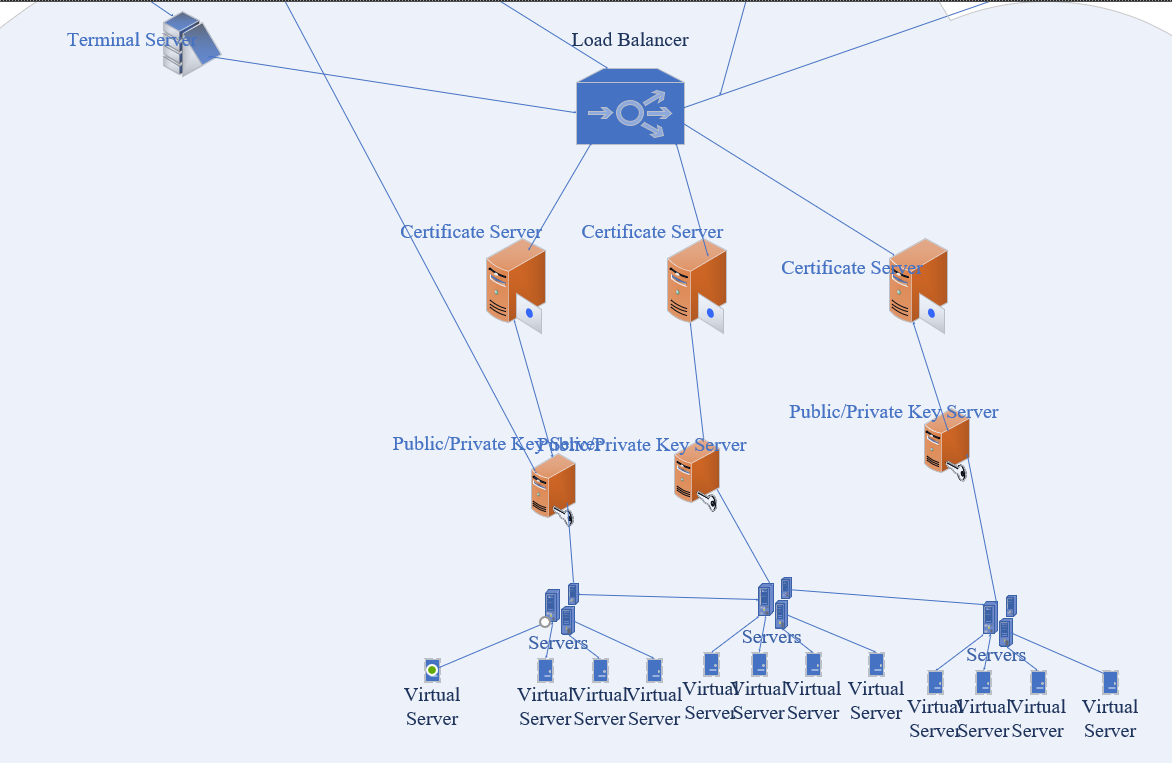
BSA425

**System Diagrams**

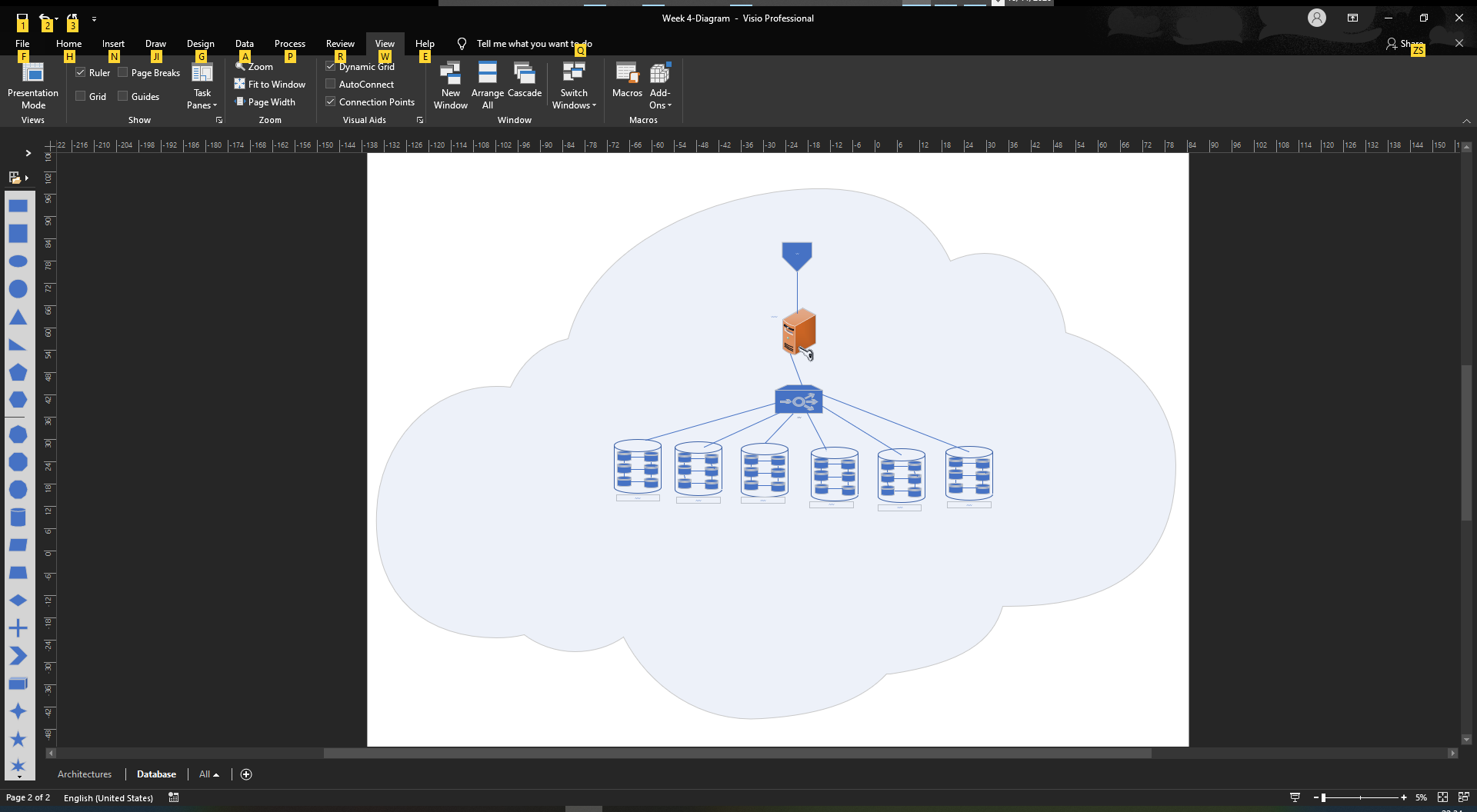
AWS built infrastructure within a cloud framework



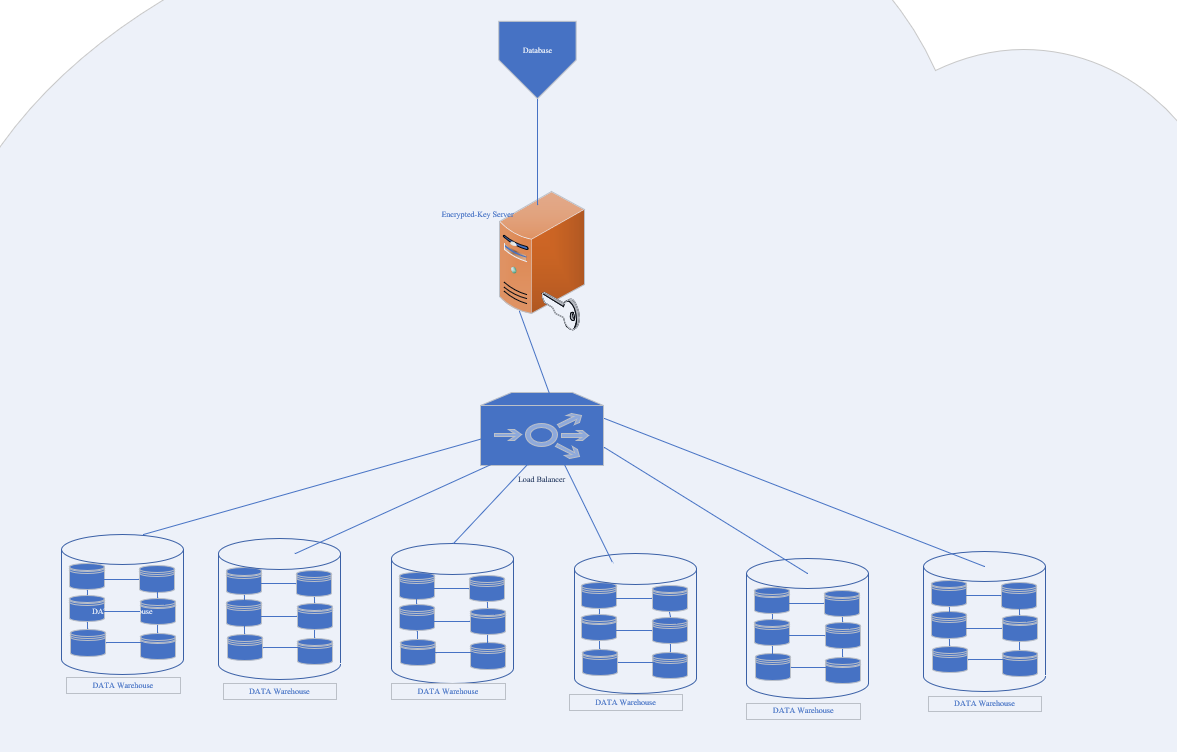
Close of view of cloud architecture; use of secures servers, terminal servers, load balancers, and virtual machines in order to power Mojo’s system.



Database-System as part of cloud configuration.

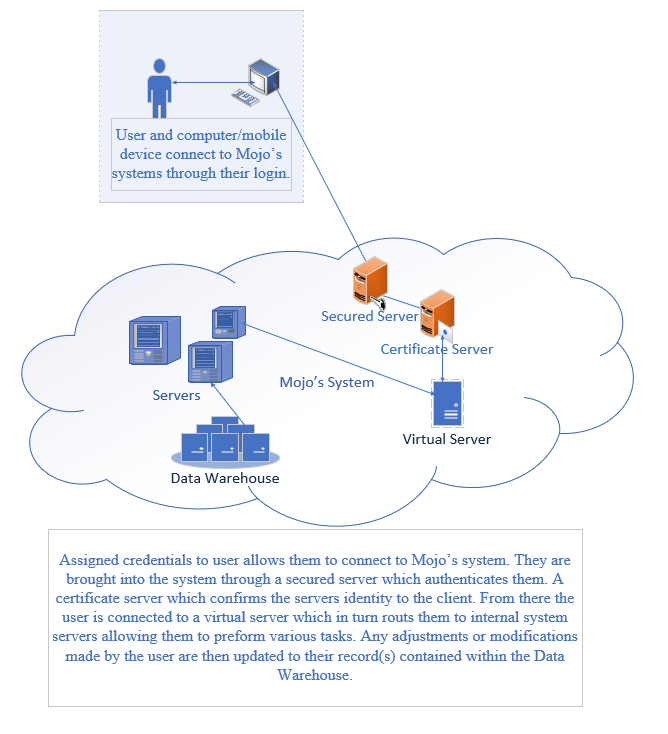


Database-System upclose view

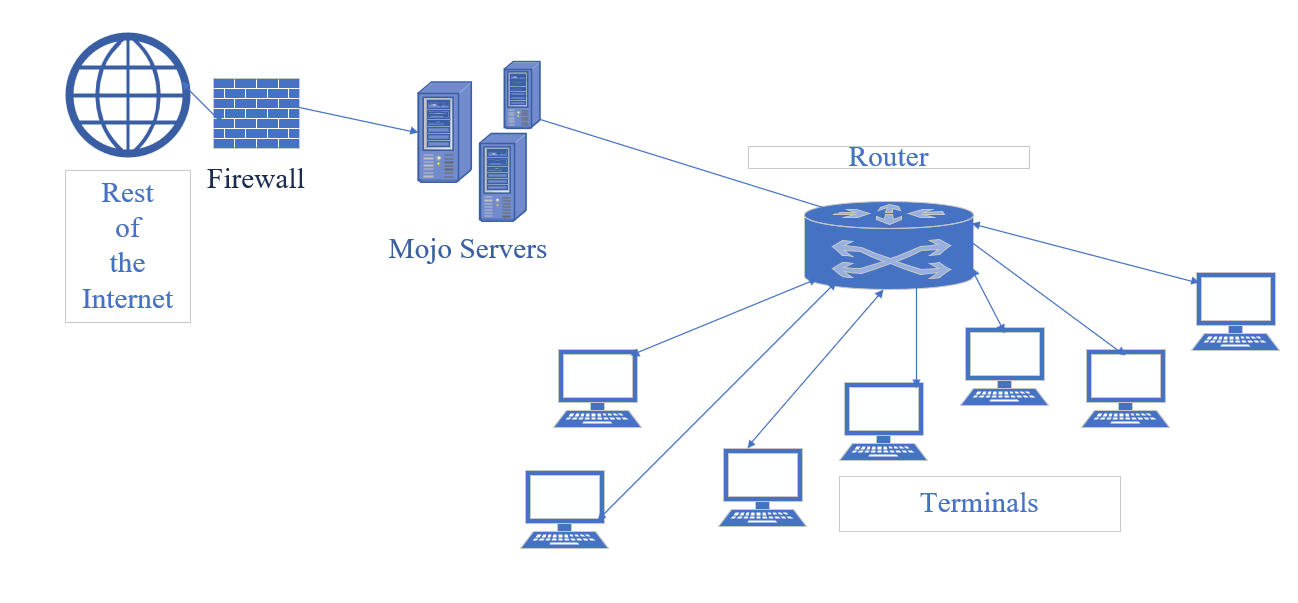


Network Diagram

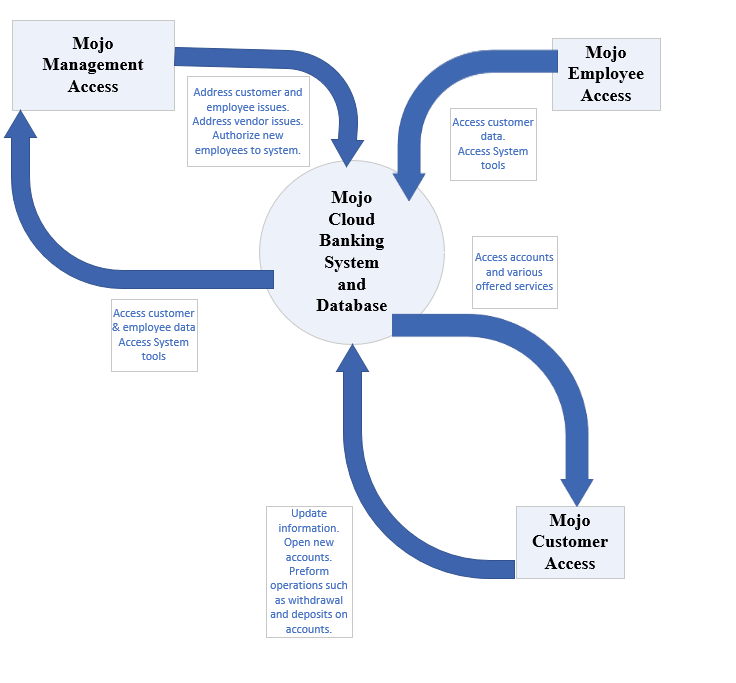
**Dataflow Diagram**



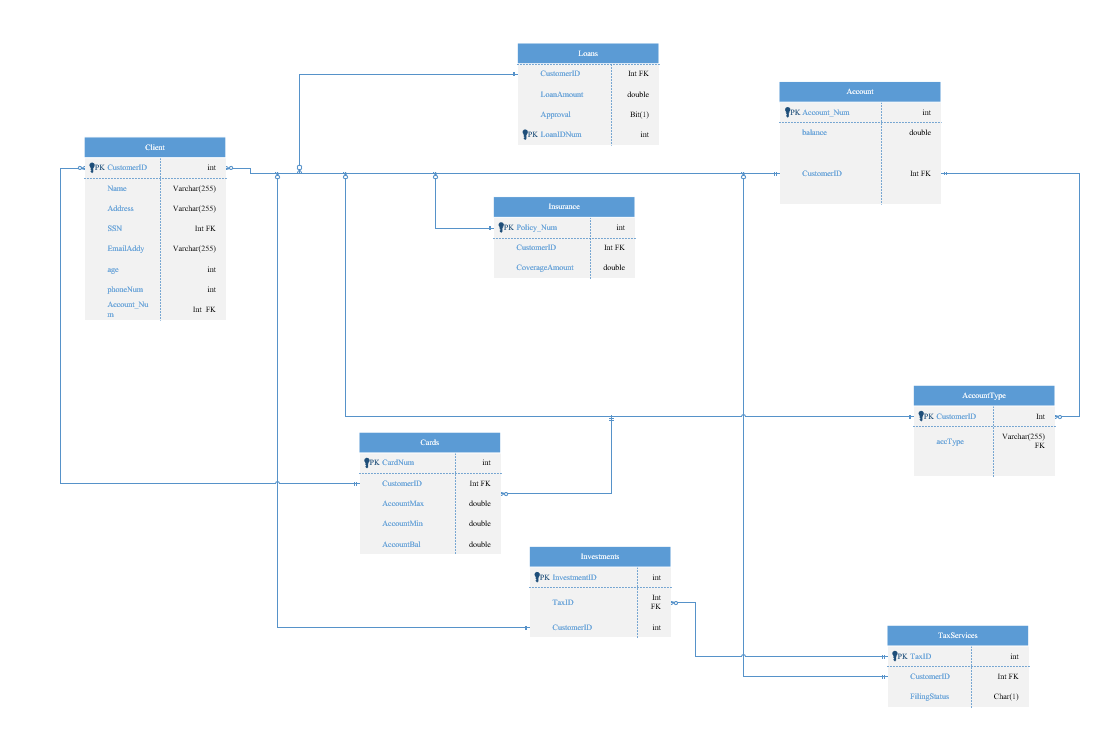
**Network Diagram**



**System Context**



**Entity-Relationship Diagram**



**Data Dictionary**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Name** | **Data Abbreviated** | **Data Type** | **Reason for Data** | **Primary Key, Foreign Key, Neither** | **Tables Present In** |
| Name | Name | Varchar | Stores customer name database | Neither | Client |
| Customer Identification | CustomerID | Int | Unique reference number for customer | Primary Key/Foreign Key | Client, Cards, Investments, Account Type, Account, Loans |
| Address | Address | Varchar | Stores customers address info | Neither | Client |
| Social Security Number | SSN | Int | Stores customers SSN | Neither | Client |
| Electronic Mail | EmailAddy | Varchar | Stores customers email | Neither | Client |
| Age | age | Int | Stores customers age | Neither | Client |
| Phone Number | PhoneNum | Int | Stores customers phone number | Neither | Client |
| Account Number | AccountNumber | Int | Stores account number | Foreign Key/Primary Key | Client, Account |
| Policy Number | Policy\_Num | Int | Stores policy number (insurance) | Primary Key | Insurance |
| Coverage Amount | CoverageAmount | Double | Stores coverage amount | Neither | Insurance |
| Approval | Approval | Bit | Used to determine loan approval through 1 and 0 bits. | Neither | Loans |
| Loan Identification | LoanIDNum | Int | Unique reference for loan | Primary Key | Loans |
| Tax Identification | TaxID | Int | Unique reference for tax filing | Primary Key/Foreign Key | TaxServices |
| FilingStatus | FilingStatus | Char | Stores value to indicate single or married filing status. | Neither | TaxServices |
| Account Type | accType | Varchar | Stores what kind of account it is. | Neither | AccountType |
| Card Number | CardNum | Int | Unique Identification for issued card | Primary Key | Cards |
| Account Maximum | AccountMax | double | Stores maximum account value | Neither | Cards |
| Account Minimum | AccountMin | double | Stores minimum account value | Neither | Cards |
| Account Balance | AccountBal | double | Stores current account balance | Neither | Cards |

The data itself is broken up into the various respective tables. It is then sorted and retrieved by its respective primary and foreign keys. By the linkage of primary and foreign keys one can call up various tables or all together using basic SQL commands. One of the most common linking data piece is CustomerID as this is a unique variable that links the customer table to the other tables present within the database itself.

**Cybersecurity Plan**

First off there would be the company itself. Within the company would be the CEO, CFO, CSO, CISO, and all its employees both bottom tier and management alike. This would also leave any investors and clients. This would also be Amazon Web Services (AWS) who’s infrastructure is being utilized to build and maintain the infrastructure of Mojo’s online based banking service being provided.

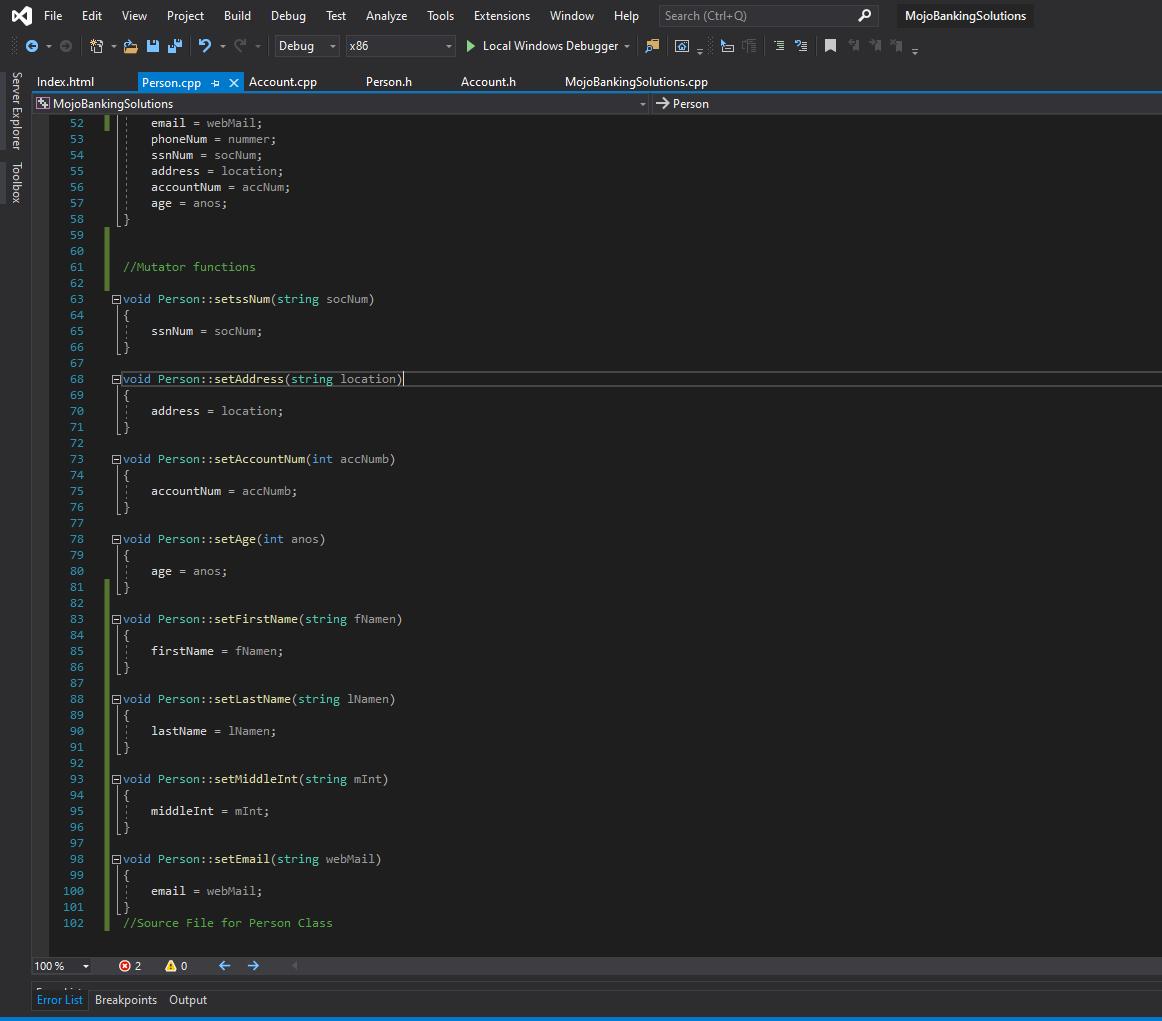
Threats can come in all shapes in sizes. This could be a Distributed Denial of Service (DDoS) Attack, Malware attacks, Phishing, and so on. How does one deal with these? As the adage goes, “the best defense is a good offence”. There are a multitude of means by which can protect Mojo from such attacks. The current standard is 128 bits of encryption that is standard use to keep bank websites secure. This level of encryption is necessary in order to protect any data being sent to or received from servers between the company and its users. In order to mitigate possible DDoS attacks the logical approach is to over-provision bandwidth, this will absorb DDoS bandwidth peaks as well as utilize software-based scanning tools to monitor network traffic and detect malicious users (Imperva, 2020). AWS has a its Shield tool allows the safeguard of running applications within AWS structures with always-on detection and automatic inline mitigations that minimalize application downtime and latency (AWS, 2020). AWS’s shield also offers a Threat landscape report (TLR) (Pinho, 2020). A part of the TLR allows one to adjust AWS’s Managed Rules, this allows for permissions and control within an AWS driven application (such as Mojo’s services) that in turn can be automated in order to allow one to simply set the appropriate parameters and have the system monitor for any indications of any traffic or invasion within the system outside the established parameters, from here the appropriate personnel can be notified (West, 2019).

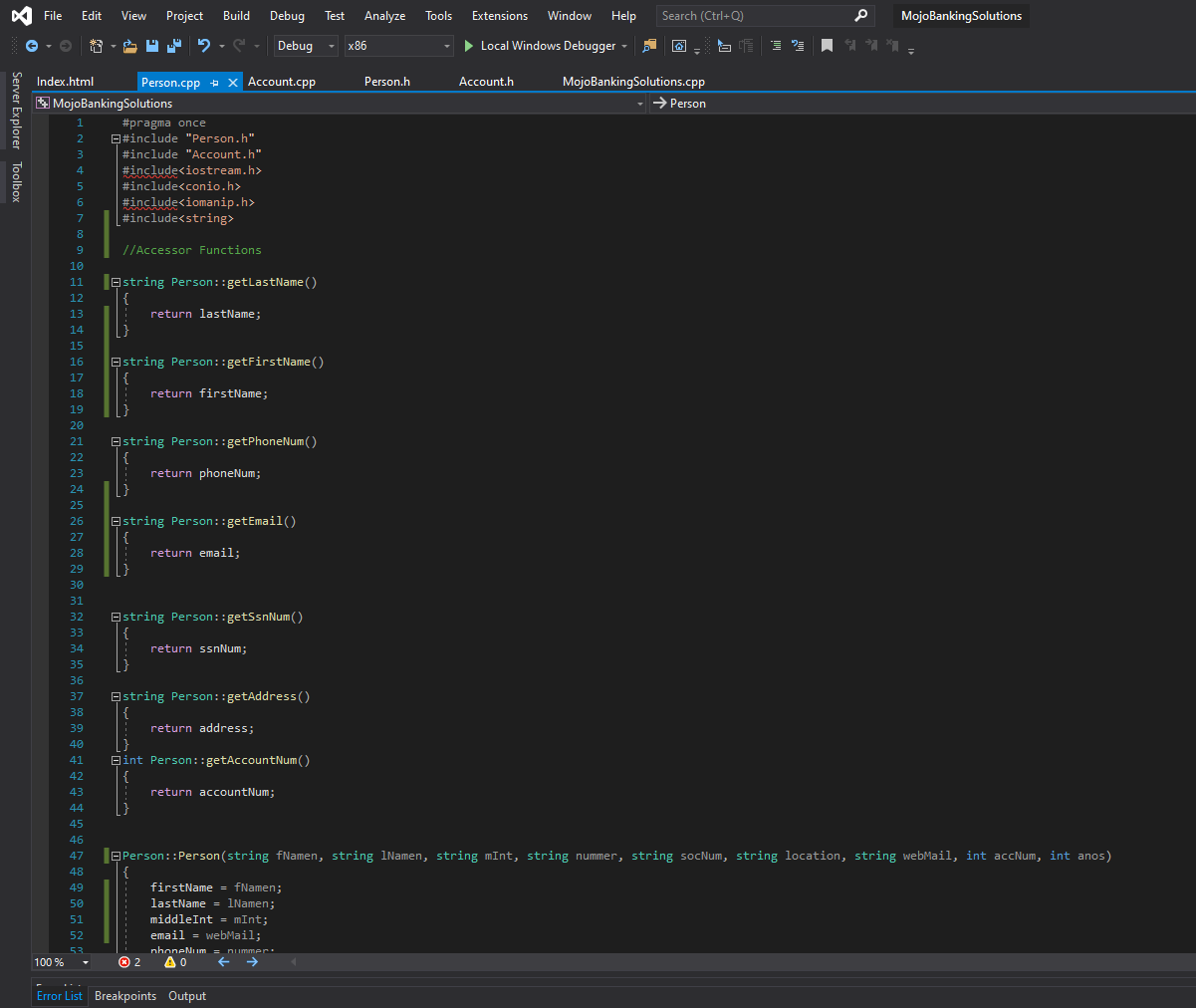
There is also the establishment of security policies. Security should never be the sole responsibility of any one entity within an organization. Logically Mojo’s IT department will handle any IT issues as well as security related issues for IT, but it is the responsibility of every stakeholder to adhere to security policies. These policies will involve the training of personnel in the basic response to security threats such as phishing attacks and how such attacks can and will manifest themselves. Documentation shall be generated indicating the proper use of company resources as well as other policies including but not limited to passwords, email and communication handling, and access protocols. The concept of least privilege will be applied to every credentialed user alone with multi-factor authentication; simply one will be provided the means to carry out their appropriate tasks within Mojo’s systems based on need. Multifactor authentication will also help to ensure the mitigation of any encroachment through users’ logins both for employees of Mojo and for customer’s in their entirety. Routine training will be established to also educate employees on current safe practices as they handle and utilize Mojo’s systems.

With the utilization of AWS Inspector one can establish automated security assessments that help improve security and compliance of all of one’s applications deployed on AWS. Combined with periodic security checks this can help to ensure all systems of Mojo remain secure at acceptable risk levels (AWS, 2020).

**Source Code**

The following is a basic bank account and person class implemented in language C++. The intention for which is to act as the backend support for Mojo Banking Solutions, Inc. web-based interface. The implementation of which was going to be combined with a Node.JS, C++ itself would run the backend implementation of the functionality tools of Mojo Banking Solution, Inc.’s webpage while JavaScript controls the front-end aspects.

**Person Class**



#pragma once

#include "Person.h"

#include "Account.h"

#include<iostream.h>

#include<conio.h>

#include<iomanip.h>

#include<string>

//Accessor Functions

string Person::getLastName()

{

return lastName;

}

string Person::getFirstName()

{

return firstName;

}

string Person::getPhoneNum()

{

return phoneNum;

}

string Person::getEmail()

{

return email;

}

string Person::getSsnNum()

{

return ssnNum;

}

string Person::getAddress()

{

return address;

}

int Person::getAccountNum()

{

return accountNum;

}

Person::Person(string fNamen, string lNamen, string mInt, string nummer, string socNum, string location, string webMail, int accNum, int anos)

{

firstName = fNamen;

lastName = lNamen;

middleInt = mInt;

email = webMail;

phoneNum = nummer;

ssnNum = socNum;

address = location;

accountNum = accNum;

age = anos;

}

//Mutator functions

void Person::setssNum(string socNum)

{

ssnNum = socNum;

}

void Person::setAddress(string location)

{

address = location;

}

void Person::setAccountNum(int accNumb)

{

accountNum = accNumb;

}

void Person::setAge(int anos)

{

age = anos;

}

void Person::setFirstName(string fNamen)

{

firstName = fNamen;

}

void Person::setLastName(string lNamen)

{

lastName = lNamen;

}

void Person::setMiddleInt(string mInt)

{

middleInt = mInt;

}

void Person::setEmail(string webMail)

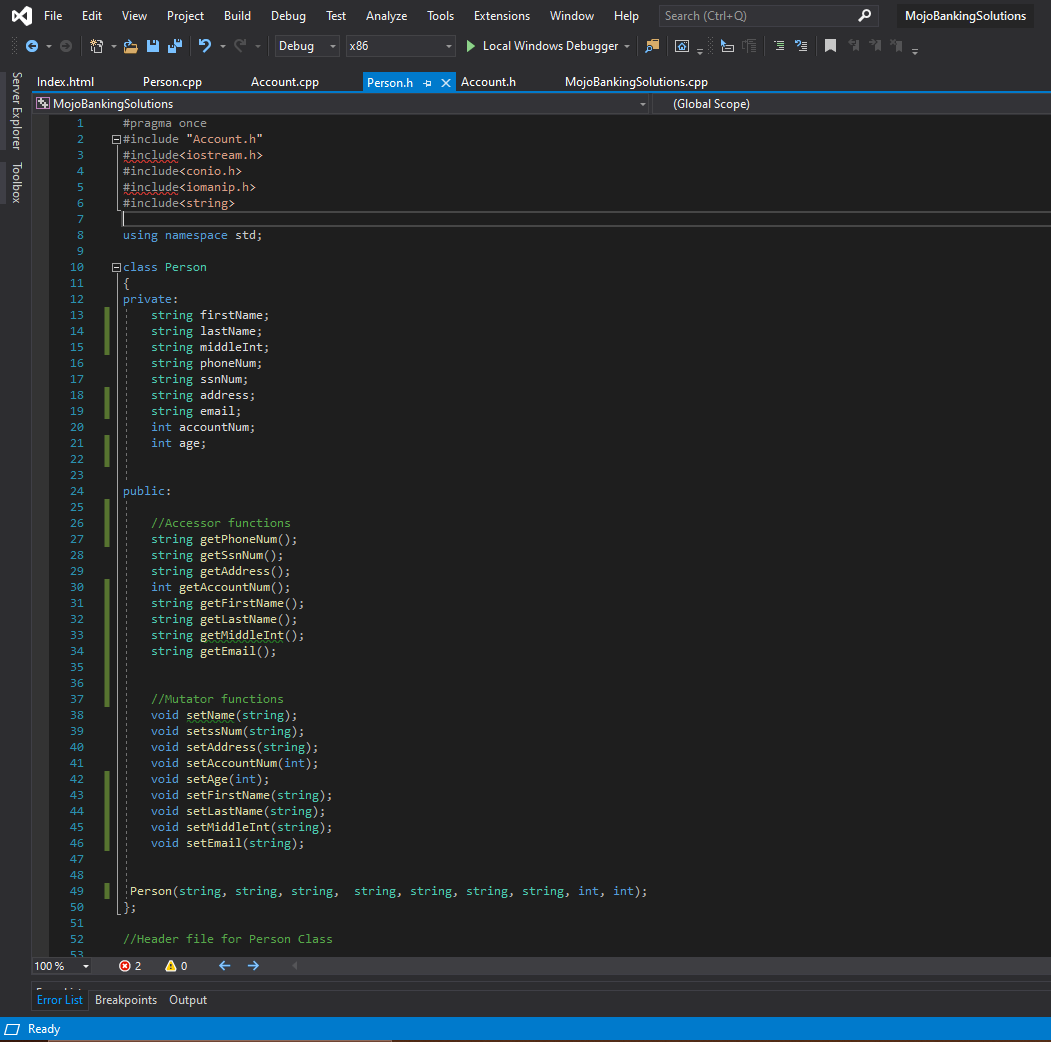
{

email = webMail;

}

//Source File for Person Class

**Person Header**



#pragma once

#include "Account.h"

#include<iostream.h>

#include<conio.h>

#include<iomanip.h>

#include<string>

using namespace std;

class Person

{

private:

string firstName;

string lastName;

string middleInt;

string phoneNum;

string ssnNum;

string address;

string email;

int accountNum;

int age;

public:

//Accessor functions

string getPhoneNum();

string getSsnNum();

string getAddress();

int getAccountNum();

string getFirstName();

string getLastName();

string getMiddleInt();

string getEmail();

//Mutator functions

void setName(string);

void setssNum(string);

void setAddress(string);

void setAccountNum(int);

void setAge(int);

void setFirstName(string);

void setLastName(string);

void setMiddleInt(string);

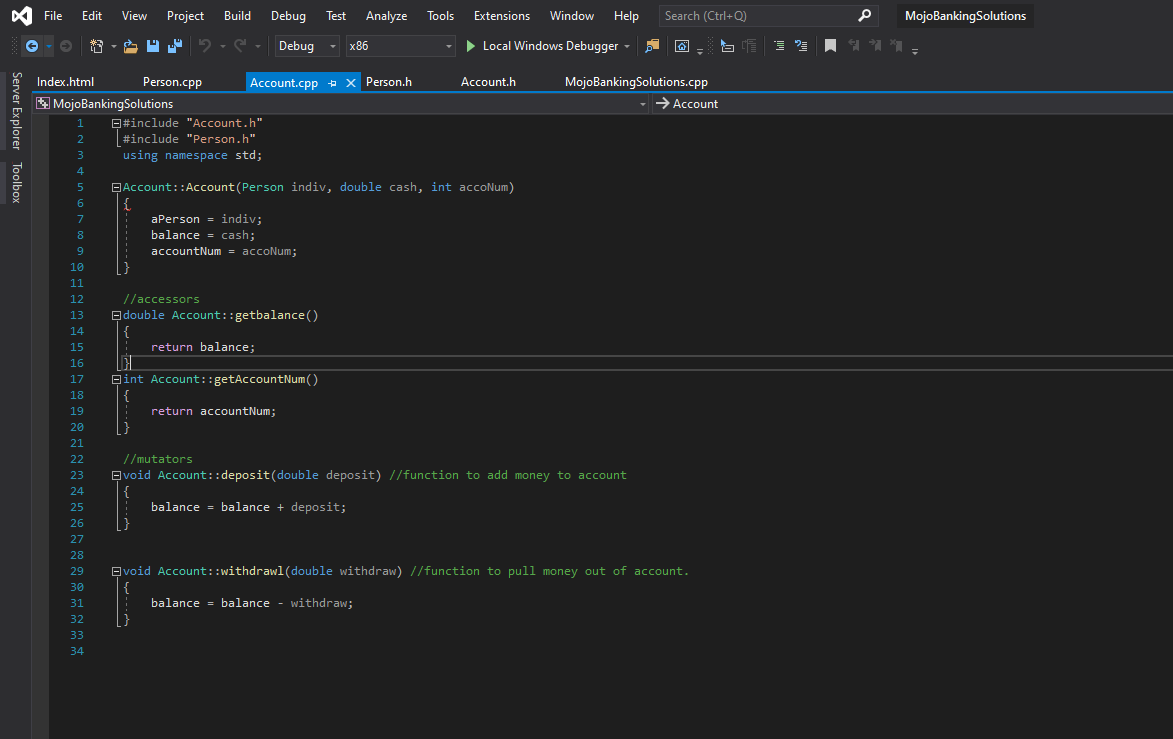
void setEmail(string);

Person(string, string, string, string, string, string, string, int, int);

};

//Header file for Person Class

**Account Class**



#include "Account.h"

#include "Person.h"

using namespace std;

Account::Account(Person indiv, double cash, int accoNum)

{

aPerson = indiv;

balance = cash;

accountNum = accoNum;

}

//accessors

double Account::getbalance()

{

return balance;

}

int Account::getAccountNum()

{

return accountNum;

}

//mutators

void Account::deposit(double deposit) //function to add money to account

{

balance = balance + deposit;

}

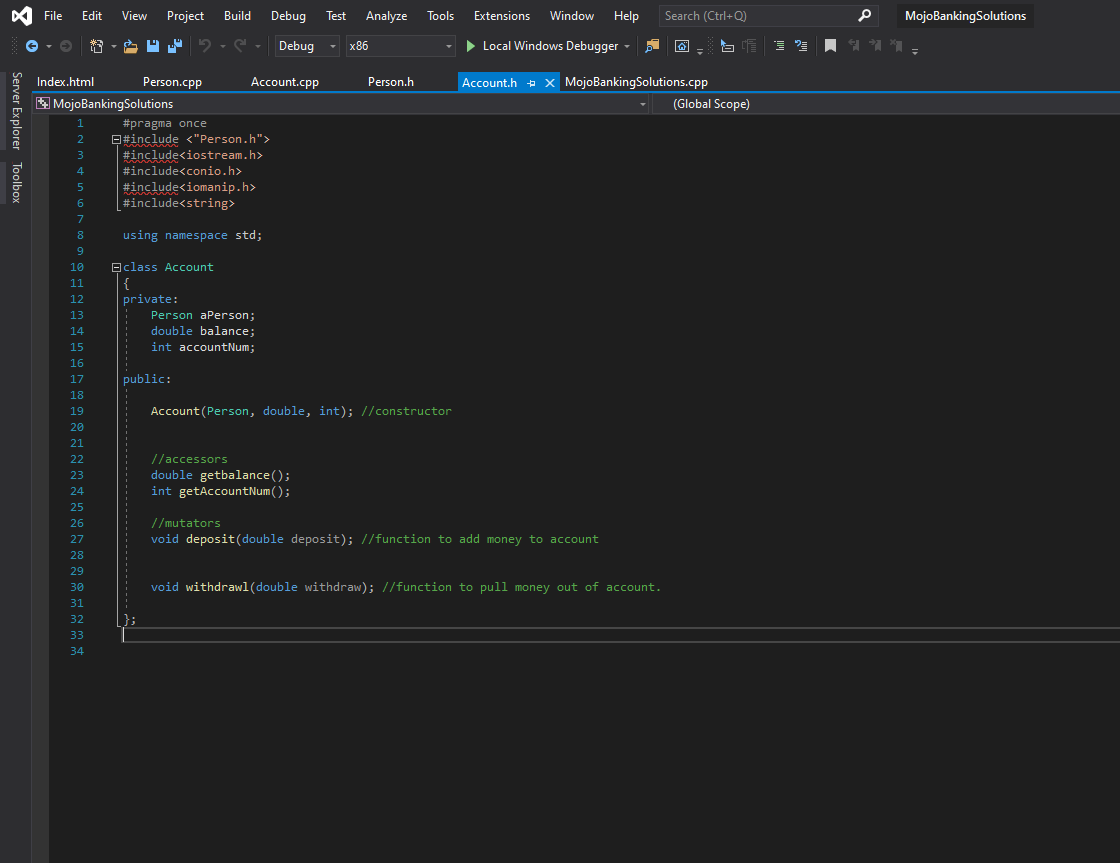
void Account::withdrawl(double withdraw) //function to pull money out of account.

{

balance = balance - withdraw;

}

**Account Header**



#pragma once

#include <"Person.h">

#include<iostream.h>

#include<conio.h>

#include<iomanip.h>

#include<string>

using namespace std;

class Account

{

private:

Person aPerson;

double balance;

int accountNum;

public:

Account(Person, double, int); //constructor

//accessors

double getbalance();

int getAccountNum();

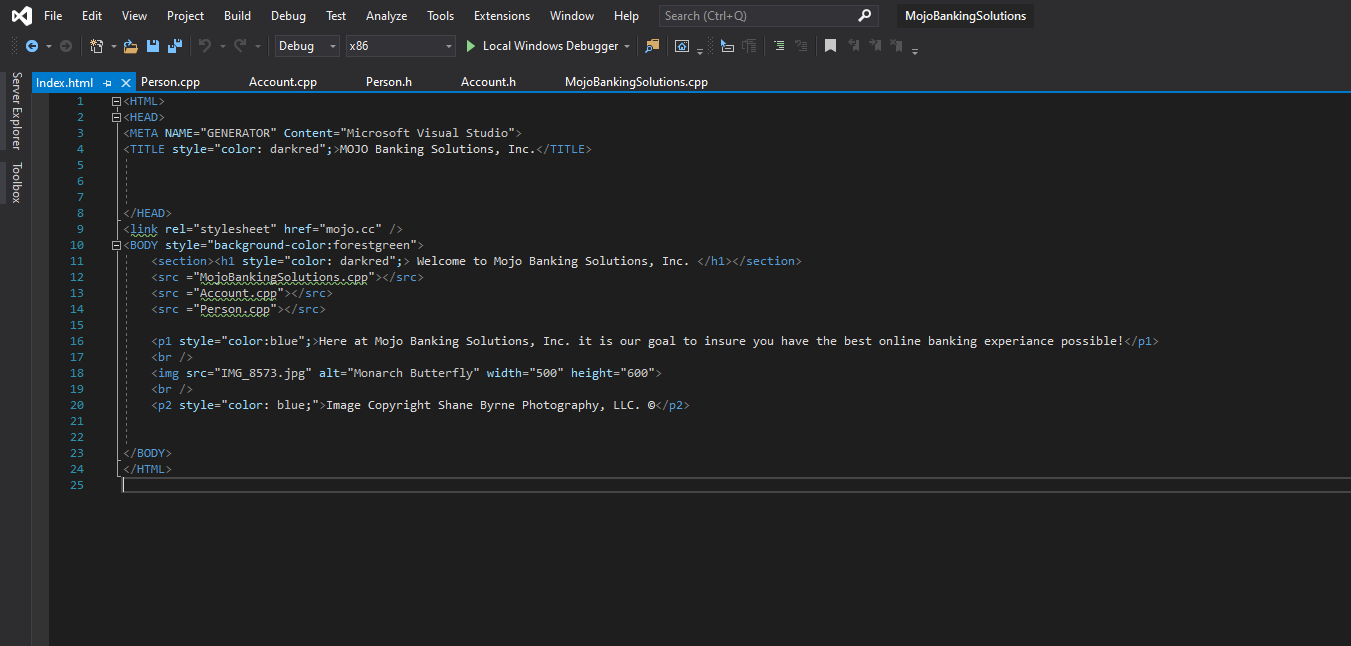
//mutators

void deposit(double deposit); //function to add money to account

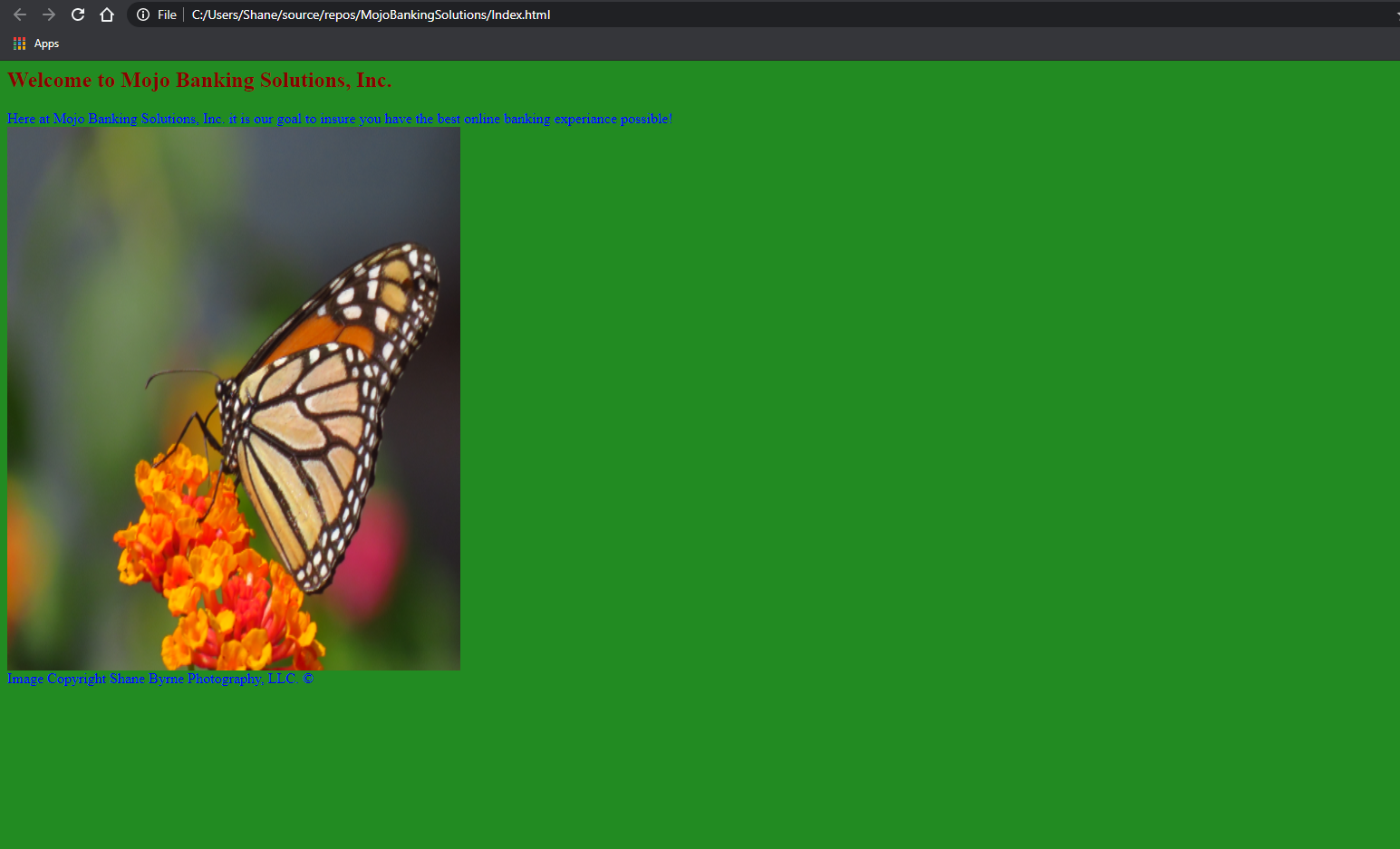
void withdrawl(double withdraw); //function to pull money out of account.

};

**Mojo Banking Solutions, Inc. Index**



**Mojo Banking Solutions, Inc. Index (Browser)**



**Mojo Banking Solutions, Inc. HTML Code**

<HTML>

<HEAD>

<META NAME="GENERATOR" Content="Microsoft Visual Studio">

<TITLE style="color: darkred";>MOJO Banking Solutions, Inc.</TITLE>

</HEAD>

<link rel="stylesheet" href="mojo.cc" />

<BODY style="background-color:forestgreen">

<section><h1 style="color: darkred";> Welcome to Mojo Banking Solutions, Inc. </h1></section>

<src ="MojoBankingSolutions.cpp"></src>

<src ="Account.cpp"></src>

<src ="Person.cpp"></src>

<p1 style="color:blue";>Here at Mojo Banking Solutions, Inc. it is our goal to insure you have the best online banking experiance possible!</p1>

<br />

<img src="IMG\_8573.jpg" alt="Monarch Butterfly" width="500" height="600">

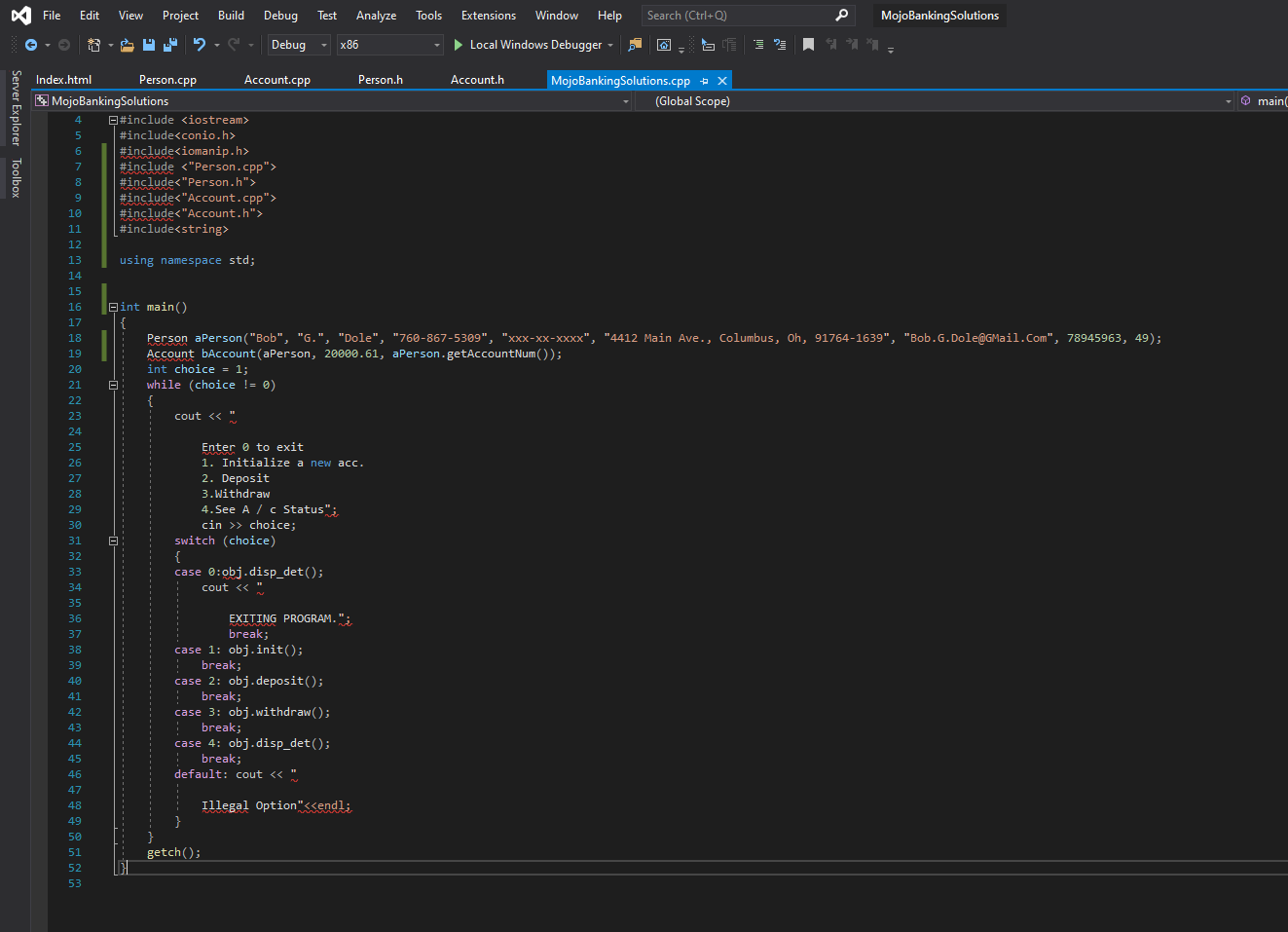
<br />

<p2 style="color: blue;">Image Copyright Shane Byrne Photography, LLC. ©</p2>

</BODY>

</HTML>

**MojoBankingSolutions.cpp (Test file)**



**MojoBankingSolutions.cpp (Test file) Code**

// MojoBankingSolutions.cpp : This file contains the 'main' function. Program execution begins and ends there.

//

#include <iostream>

#include<conio.h>

#include<iomanip.h>

#include <"Person.cpp">

#include<"Person.h">

#include<"Account.cpp">

#include<"Account.h">

#include<string>

using namespace std;

int main()

{

Person aPerson("Bob", "G.", "Dole", "760-867-5309", "xxx-xx-xxxx", "4412 Main Ave., Columbus, Oh, 91764-1639", "Bob.G.Dole@GMail.Com", 78945963, 49);

Account bAccount(aPerson, 20000.61, aPerson.getAccountNum());

int choice = 1;

while (choice != 0)

{

cout << "

Enter 0 to exit

1. Initialize a new acc.

2. Deposit

3.Withdraw

4.See A / c Status";

cin >> choice;

switch (choice)

{

case 0:obj.disp\_det();

cout << "

EXITING PROGRAM.";

break;

case 1: obj.init();

break;

case 2: obj.deposit();

break;

case 3: obj.withdraw();

break;

case 4: obj.disp\_det();

break;

default: cout << "

Illegal Option"<<endl;

}

}

getch();

}

**Quality Assurance & Software Test Plan**

**Software Test Plan**

Mojo Banking Solutions, Inc..

The project in and of itself is that of an online based bank which provides various services to its customers such as accounts setup, deposits, credit debit and travel cards. It will provide loans, insurance, investments, and tax services. This service will provide the scalability, availability, security, and manageability one would expect from an online based banking service both through its web-based interface as well as mobile application-based interface.

# Features To Be Tested / Not To Be Tested

|  |  |
| --- | --- |
| **Features to be tested** | **Features to not be tested** |
| View account summary | Interface customization: Not a mission critical component, if there are issues with this it can be addressed through patching. |
| View/Print statements |  |
| Set up online payments. |  |
| Reorder cards (debit, credit, travel) |  |
| Transfer funds between accounts (i.e. checking to savings or visa versa) |  |
| Access other accounts (CD's, IRA's, etc.) |  |
| Set up different accounts (savings, checking, CD's, IRA's, etc.) |  |
| Update personal information (addresses and phone numbers for example |  |
| Set/Change password |  |
| Biometric identification (mobile application) |  |
| View history |  |
| Set/change user id |  |
| Enable/disable account alerts (e-mail/SMS). |  |
| Set up customed alerts (account balance for example) |  |
| Withdrawn and deposit function accuracy (make sure functions are executing properly. |  |
| User ID retrieval |  |

# Testing Pass / Fail Criteria

* Suspension criteria, this would be any situation which impeded the ability to continue testing or value in preforming testing that leads to the suspension of testing activities.
* Resumption criteria, this being a continuation of testing after the issues causing the suspension criteria to have been corrected and patched accordingly.
* Approval criteria, this will be a part of the application being tested that has performed its desired functionality
* Unit testing, each unit should be tested under its designed function, if unit can not live up to its expectations a fail condition is to be issued and further adjustment to its source code and algorithm shall be made.
* Data leakage, test should be run to ensure the source code fallows its constraints as per the programming principal of encapsulation. This would mean one not having the capacity to interact with nor alter the data within the confines of the program. Fail condition would be one being able to access and/or alter the data from the program’s interfaces given.

# Testing Approach

. Initial stage will be actual planning of the tests in their entirety. This will bring in the scope of the test itself, the risks associated with each test, and the effect of the development of the entire software package is determined, as well as the implications given the impact analysis of the application being tested.

The second stage is creation of scenarios to test various parts of the application. Primary testing should be done on critical systems especially any parts of the application dealing with finances and any interaction with internal or external systems. Functionality checks should be made on any units within the application that deal with any calculation functions especially those that pertain to financial information (example being account balances). Next should be the behavior of the code, an example being the use of CSS established keywords and how there should be a uniform response with their utilization within the webpage (i.e. H1 tags labeled as “red” in CSS should display as “red” when executed). Functionalities should be tested on page functions as well (input and output, delete functions and so on for example).

Next should be a constructed environment to perform an execute tests in. This should be isolated from any other systems to eliminate any possible outside interference until the units being tested are passed and ready for integration. Generic falsified data should be generated in order for it to be utilized in the testing process. Each unit should be subjected to the appropriate data (mock financial data for financial parts of the application for example). Any failures of a unit or part of the system should be identified and corrected before the integration testing.

Next the scenarios shall be running the scenarios. Adequate time should be scheduled to elevate and time pressure for the testers. The fully planned scenarios should be executed in several iterations until the part of application itself is behaving as expected.

Finally, all results should be recorded and documentation. This report should be simplified in presentation containing screenshots or video documentation of any encountered errors as well as documentation of how the error was addressed and resolved (if possible).

Testing methods utilized shall be unit testing, integration testing, system testing, and acceptance testing. Unit testing itself will be the handling of partitions of the application itself, the application in its entirety shall be broken up into smaller unites to be tested (financial partitions tested together for example). The core idea behind unit testing is to test the smallest piece of code, in this instance it would be the various classes derived from C++. From here automated tools can ensure the code in question is capable of being automated and easily operated repeatedly in succession. Next would be integration testing, this would be the combining of different units into a conglomeration of blocks resembling functions of the overall system. In system testing (or end-to-end testing) would be a full test of the application. After all integrated blocks are integrated themselves into the complete application system testing itself would be applied to the whole of the application to ensure all its functions and devised tools built into the application are functionally working as designed. Finally, acceptance testing is the testing of the application form the end-user perspective. This is to ensure the built-in tools allowing the user to interact with the application are in fact alloying the appropriate conditions and actions.

# Testing Cases

|  |  |
| --- | --- |
| **Test Case** | **Test Scenario** |
| 1 | Check system response on entering valid username/password |
| 2 | Check system response on entry of invalid username/password. |
| 3 | Check system response on entry of no entry of username or password. |
| 4 | Check system response on the update of user data. |
| 5 | Check system response of account information access handling tools such as viewing and adding to one's balance for example. |
| 6 | Check system response for computational functions. |
| 7 | Check system response for database update. |
| 8 | Check system response for instanciation creation from functions (accounts). |
| 9 | Check system response for switching to different user functions (Taxes, insurance, etc.) |
| 10 | Check system response for notifications (e-mail & SMS) |
| 11 | Check system response for custom notifications (Account balance for example). |
| 12 | Check system response for duplicate account creation (duplicate username creation for example) |

# Testing Materials (Hardware/Software Requirements)

Materials that are needed for the testing of an application being the actual environment the application. This would be the physical servers being used as well as an array of generated virtual servers the application would depend on. A functioning Datawarehouse would be best to test the retrieval and placement of data being entered into the system itself through its constructed interfaces. An Interactive Development Environment would have to be loaded onto a host machine and configured to work build C++ code structures into for the beginning of said project. Microsoft’s Visual Studio Code is easy to set up and easy to use for such a venture as an example. While as the front end application of the system is ran using JavaScript, programs such as MochaJS can be used in building that respective code, just as with Visual Studio it would simply need a host machine to operate the code on and trouble shoot it. Terminals would be needed to connect to the cloud infrastructure as laid out to access systems. While as the system itself would exist in a cloud environment, one would still need an office to operate out of at least while the application is being tested, this would allow easier collaboration for the build team. One would also need an entire testing tool such as Clicqa Bank Testing Services in order to test the entirety of the banking application.

# Testing Schedule

| Testing Activity | Duration | Resource | Comments |
| --- | --- | --- | --- |
| Test Plan Creation | 5 days | Test Manager | Establishment of test plan protocols. |
| Test Specification Creation | 10 days | Test Leads | Establishment of testing specifications and the criteria the testing team will have to follow. |
| Test Specification Team Review | 5 days | Project Team | Collaboration of the testing team in order to review testing guidelines, policies, and factors. |
| Component Testing | 20 days | Component Testers | Testing of each constructed segment of the application. |
| Integration Testing | 20 days | Component and System Testers | Testing of each constructed after it is integrated with other segments. |
| System Testing | 15 days | System Testers | Testing of the entire system application after all parts have been integrated. |
| Performance Testing | 5 days | System Testers | Putting the system under different constraints to ensure it can handle workload and live up to established expectations. |
| Use Case Validation | 10 days | System Testers | Testing the system from the customers perspective to ensure all desired functionalities are met. |
| Alpha Testing | 5 days | Product Managers / Analysts | Testing the system in house to ensure each system behaves as desired for the companies side of the application as well as the end-user or customers side of the application. |
| Beta Testing / Pilot Program | 20 days | Pilot Program End-Users | Testing the system just before launch in its cloud environment before allowing outside user (customer) access. |

# Risks and Contingencies Matrix

| Risk | Probability | Risk Type | Owner | Contingencies / Mitigation Approach |
| --- | --- | --- | --- | --- |
| Do not have enough skilled workers to test components as they are ready for testing. | 25% | Project Resources | Testing Manager | Testing schedule will be adjusted based upon available resources. |
| Testing team member turnover | 10% | Project Resources | Testing Manager | Adjust testing schedules. Make sure testing team members are cross-trained on testing techniques in case a team member leaves the organization. |
| Having enough infrastructure to support rollout demands | 10% | Project Resources | Project Supervisor | Adequately building needed infrastructure to successfully test the application under stressed conditions. |
| Scope Creep | 25% | Project Resources | Testing Manager | Ensuring application is tested within the established parameters needed. |
| Scheduling deviation | 15% | Project Resources | Testing Manager | Ensuring proper testing protocols are called out in their scheduled allotted time. |

**References**

AWS (2020) AWS Inspector Retrieved from https://aws.amazon.com/inspector/

AWS (2020) AWS Shield Retrieved from <https://aws.amazon.com/shield/?whats-new-cards.sort->

by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc

Choudry, A. (May 14, 2020) How to Write a Good Test Case in Software Testing?

Retrieved from https://www.edureka.co/blog/test-case-in-software-testing/

Delmater, M., & Murach, J. (2018). Murach's C++ Programming . Mike Murach & Associates, Inc..

Erikson, U. (16th September 2016) Top 10 Attributes Every Sample Test Plan Document Must Have Retrieved from

https://reqtest.com/testing-blog/sample-test-plan/

Impervia (2020) Protecting Financial Institution from DDoS Attacks Retrieved from

<https://www.imperva.com/resources/resource-library/white-papers/protecting-financial-institutions-ddos->

attacks-wp-ty?lang=EN&asset\_id=1252

Kenny, P. (2020) Common Online Banking Features Retrieve from

https://www.streetdirectory.com/travel\_guide/145303/banking/common\_online\_banking\_features.html

Marsh, D. (February 13, 2017) Pass/Fail Criteria Retrieved from <https://sites.psu.edu/softwarequalityassurance/passfail->

criteria/

MyITpros (2020) How to create a simple cybersecurity plan for your business Retrieved from

https://www.myitpros.com/myitpros-blog/how-to-create-a-simple-cybersecurity-plan-for-your-business

Ӧztürk, M. (Jan 4 2020) Software Testing Processes and Levels of Testing Retrieved from

https://medium.com/swlh/software-testing-process-and-levels-of-testing-4274904ce655

Phino, M. (29 May, 2020) AWS Shield Threat Landscape report is now available Retrieved from

https://aws.amazon.com/blogs/security/aws-shield-threat-landscape-report-now-available/

Software Testing Help (September 1, 2020) 180+ Web Application Testing Example Test Cases (Sample Checklist)

Retrieved from https://www.softwaretestinghelp.com/sample-test-cases-testing-web-desktop-applications/

Stevens, A., & Walnum, C. (2000). Standard C++ Bible . IDG Books Worldwide, Inc..

Stroustrup, B. (2013). The C++ Programming Language (4th ed.). Pearson Education, Inc..

Stroustrup, B. (2014). Programming Principles and Practice Using C++ (2nd ed.). Person Education, Inc..

West, B. (25 Nov 201) Announcing AWS Managed Rules for AWS WAF Retrieved from

https://aws.amazon.com/blogs/aws/announcing-aws-managed-rules-for-aws-waf/