

The Key to Future Financial Management..



BONDS

The **B**ank **O**f **N**early **D**one **S**eniors

Documentation

Front End	Business Logic	Database
Brendan Beatty Brendan.Beatty@Marist.edu	Michael Anstadt Michael.Anstadt@Marist.edu	Erik Ritter Erik.Ritter@Marist.edu
Michael Guarascio Michael.Guarascio@Marist.edu	Marcelo Bajana Marcelo.Bajana@Marist.edu	Nolan Van Wert Nolan.VanWert@Marist.edu
Jake Story Jacob.Story@Marist.edu	Christopher Camp Christopher.Camp@Marist.edu	Carlton VanderWeit Carlton.VanderWeit@Marist.edu
	Bob Pelletier Robert.Pelletier@Marist.edu	
	Eric Saari Eric.Saari@Marist.edu	

Marist College
School of Computer Science and Mathematics
3399 North Road, Poughkeepsie, NY 12601

I. WHAT IS B.O.N.D.S?

B.O.N.D.S stands for Bank of Nearly Done Seniors. We are a group of seniors studying Information Technology at Marist College in Poughkeepsie, NY. Most of us will be graduating on May 21, 2005 with a Bachelors of Science degree and a minor in Information Systems. We are all highly motivated individuals and have gelled well as a team over the past Spring 2005 semester.

Theoretically what we have come together to produce is an entirely electronic banking system. B.O.N.D.S provides various services for customers and also administrative functions for employees and management on different access levels. To implement this we have broken up our team into three separate tiers.

The Front-End Coding Team consists of: Jake Story, Michael Guarascio, and Brendan Beatty.

The Back-End Coding Team consists of: Christopher Camp, Michael Anstadt, Bob Pelletier, Eric Saari, and Marcelo Bajana.

The Database Team consists of: Nolan Van Wert, Eric Ritter, and Carlton Vanderweit.

The **Front-End Coding Team** is responsible for the presentation of the website and non-server side code. The team has designed the page using DHTML, Java script and Flash in some areas. They have also designed the forms that are used by the customer to apply for an account, to apply for a loan, to check various accounts, etc. Various validations are implemented to make sure credit card data and other data is properly stored within our database.

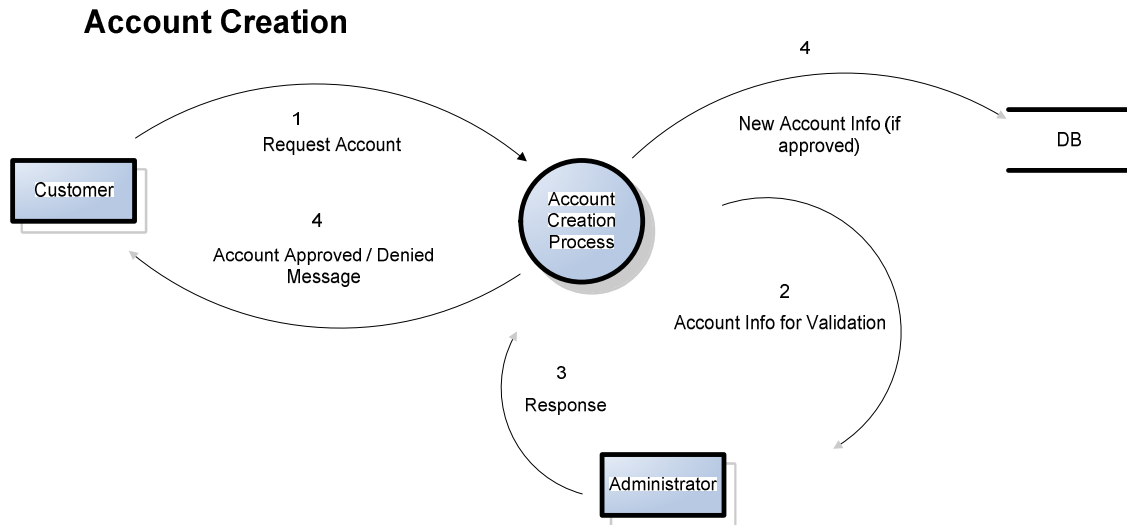
The **Back-End Coding Team** is responsible for programming the various classes and methods that provide the customers and employees that use the online bank the services they need. For example, there is a method that allows customers to deposit and withdrawal money from their various accounts. This is all done securely after a user has already registered and logged in with our web site. Most of the code is written in C# and developed in a .Net environment using Microsoft Visual Studio.

The **Database Team** is responsible for designing and fully implementing a database that supports all of the various functions we would like to support as a team. Our database is fully normalized in Boyce-Codd normal form. The database team also provides the other teams with stored procedures and triggers that are necessary to manipulate the database's data accurately and easily.

It is necessary that all three teams communicate with each other and keep each other constantly updated. Along with scheduled meetings once or twice a week we also have an emailing list that each of us are apart of so that we can easily contact each other when necessary.

II. SERVICES

The following section outlines the main services we provide on our banking web site. For each service we will include a dataflow diagram, a brief explanation of the service and a snippet of the code used to implement the service on our site.



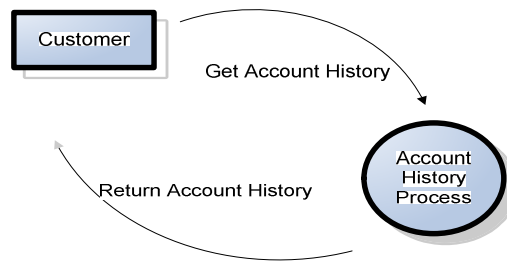
In the account creation service a customer can request an account by logging onto our web page and filling out the necessary form. A method in our back-end code stores the information in our database and marks it as pending until an administrator approves/denies the customer. A message will be sent back to the customer letting him/her know whether or not the account was approved or not.

Code Example:

```
private string CreateAccount( string AID, string OpenDate, string
Status, int AcctTypeID, double Balance)
{
    string msg = "";
    string connectStr = "server=10.128.45.26; database=bonds; user
id=bonds; password=alpaca";

    try
    {
        using (SqlConnection conn = new SqlConnection(connectStr))
        {
            conn.Open();
            if (conn.State == ConnectionState.Open)
            {
                using (SqlCommand dbCmd = new
SqlCommand())
                {
```


View Account History



A customer may want to have a record of their accounts. After logging into our system using their user ID and password they can choose to view the various accounts that they have on our database.

Code Example:

```
public string getAcctHistory (string AID)
{
    string retVal = "";
    string connectStr = "server=10.128.45.26; database=bonds;
    user id=bonds; password=alpaca";
    SqlConnection conn = new SqlConnection(connectStr);
    string query = "SELECT TID, Date, Amount, Type, q
    Description FROM Transactions WHERE aid='" + AID + "'";

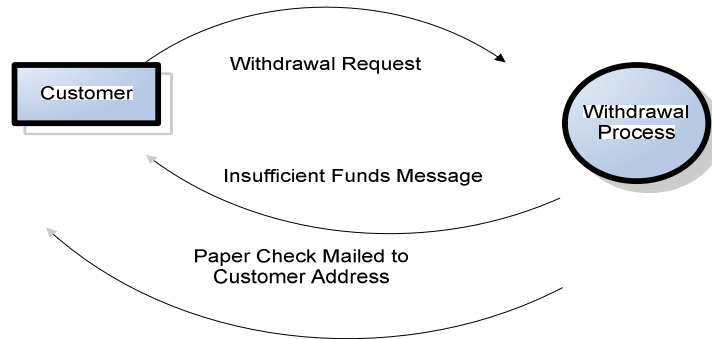
    SqlCommand statComm = new SqlCommand(query,conn);
    conn.Open();
    SqlDataReader reader = statComm.ExecuteReader();

    if(conn.State == ConnectionState.Open)
    {
        while(reader.Read())
        {
            retVal += reader.GetValue(0).ToString() + "," +
            reader.GetValue(1).ToString() + "," +
            reader.GetValue(2).ToString() + "," +
            reader.GetValue(3).ToString() + "," +
            reader.GetValue(4).ToString() + "|";
        }
    }
    else
    {
        retVal = "Could not complete transaction";
    }

    return retVal;
}
//end getAcctHistory
```

This method can also be found in the account class. It takes in the AID as a parameter. It queries the database with a SQL select command. The reader reads in the data to the code so that the front-end coding team can display the information for the customer.

Withdrawals



Some of the more common banking functions include withdrawals and deposits. Above is a dataflow diagram of the withdrawal process. A customer can request to withdrawal money from his/her account. The withdrawal process includes updating the database so that the correct balance is displayed to the customer on the webpage. If they do not have sufficient funds to withdrawal then they will receive an “insufficient funds” error message.

The screenshot shows a banking website interface. At the top, there is a banner with the text "The Key to Future Financial Management.." and a date "Wednesday, April 20, 2005". The main content area features a withdrawal form with the following fields: "amount:" with the value "100", "service:" with a dropdown menu set to "withdraw", and "transfer to:" with an empty text box. A "submit" button is located below the form. Below the form, there is a note: "Input information and click send." On the left side of the page, there is a sidebar with navigation links: "log out", "View/Edit Your Account Info", and "Deposit/Withdraw". Below these links, there is a dropdown menu for account selection, currently showing "mySavings". At the bottom of the page, there is a red navigation bar with links: "HOME | WHAT'S NEW | INVESTORS | CONTACT US | SITE MAP".

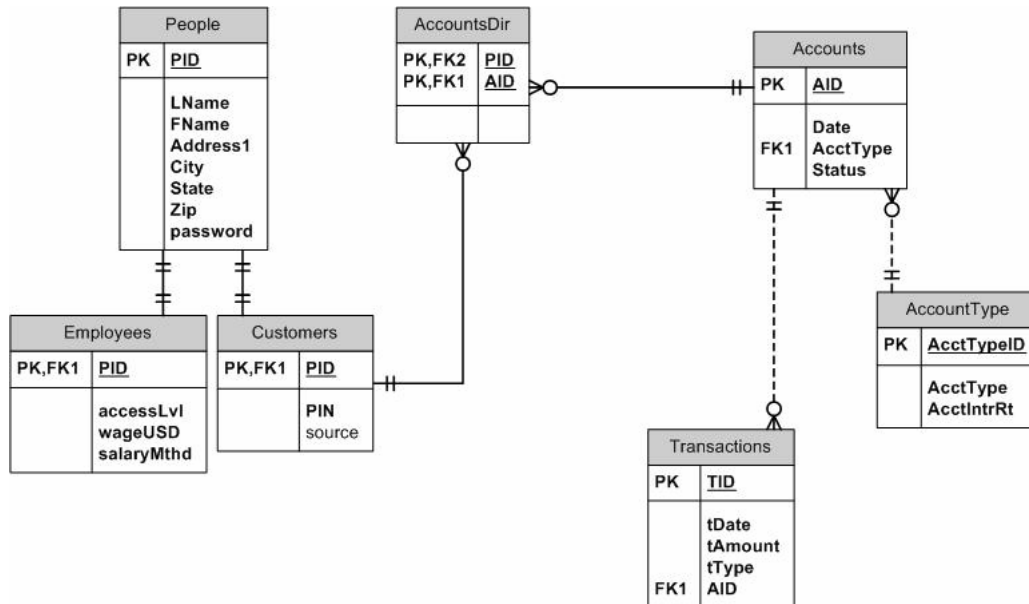
This is a screen shot of what a customer would see on our webpage when attempting to make a deposit/withdrawal. As you can see user p00001 is already logged in and ready to make a \$100 withdrawal from his/her account. When the submit button is pressed the user is brought to the following screen:

The screenshot displays a web application interface for 'BONDS Bank of Nearly Done Seniors'. The header features the slogan 'The Key to Future Financial Management..' and the date 'Wednesday, April 20, 2005'. The left sidebar contains navigation links: 'log out', 'View/Edit Your Account Info', and 'Deposit/Withdraw'. The main content area shows a withdrawal confirmation screen for user 'p00001'. The form includes fields for 'amount:', 'service:' (with a dropdown menu showing 'select service'), and 'transfer to:'. A 'submit' button is present. Below the form, a confirmation message reads: 'Thank you for your withdrawal of \$100.00. Your balance is currently \$343.00'. The footer contains navigation links: 'HOME | WHAT'S NEW | INVESTORS | CONTACT US | SITE MAP'.

The customer now receives a receipt page letting him/her know that \$100 has been successfully withdrawn from the account. There is also a line which states the current balance of the customers account.

III. DATABASE

The database team designed an Entity-Relationship Diagram. It outlines the tables in our database along with their relationship with each other. Each table has unique primary keys to differentiate each piece of data. The following is the preliminary ER Diagram.



The database team has since come together and added a Loan table to handle the added functionality we want to provide our customers. It is important that the design can adapt to change because over the course of the semester the team has come up with new ideas for the online bank.

The database team also is responsible for creating stored procedures. Stored procedures are recompiled SQL routines that are stored on the database server. Advantages of stored procedures include better performance (because they are precompiled) and reuse (since they are on the server, they can be used by anyone with access). The following stored procedure “getNextAID” returns the next highest available account ID (AID) so that the next account that is created will have a unique identification number.

Stored Procedure: getNextAID

```

Create procedure getNextAID @nextAID varchar(6) OUTPUT AS
declare @AID varchar(6)
select @AID = (Select TOP 1 AID from Accounts order by AID desc)
select @nextAID = 'A'+
CAST(Substring(CAST((CAST(((CAST(Substring(@AID,2,5) AS decimal(5,0)) +
1) / 100000)
AS decimal(5,5)))AS varchar(7)),3,5) AS varchar(5))
    
```

The following is an example of the SQL the back end coding team would use in their code in order to make use of the stored procedure:

```

declare @nextPID varchar(6)
exec getNextPID @nextPID OUTPUT
PRINT @nextPID

```

The database team also made use of triggers. Triggers are a program in a database that gets called each time a row in a table is INSERTED, UPDATED or DELETE. An example of a trigger that we implement in our database is the “updateBalance” trigger. This trigger updates the balance in the Accounts table whenever there is an insert to the Transactions table. If a withdrawal occurs, the amount is subtracted from the balance of the account with that AID otherwise it is added (deposit).

Trigger: updateBalance

```

CREATE TRIGGER updateBalance
ON Transactions AFTER Insert AS
BEGIN
declare @AID varchar(6)
declare @Amount money
declare @Type varchar(10)
select @AID = AID from inserted
select @Amount = Amount from inserted
select @Type = Type from inserted
declare @Balance money
select @Balance = Balance from Accounts Where Accounts.AID = @AID
if (@Type = 'withdrawal')
    BEGIN
        Select @Amount = @Amount * -1
    END
select @Balance = @Balance + @Amount
UPDATE Accounts SET Balance = @Balance WHERE Accounts.AID = @AID
END

```

IV. WISHLIST

In the business world it is always important to plan for the future. As a result we have devised a short wish list of functionality that we would like to include on our online bank in the future.

- We would like to include a Bill Pay function in the future which provides our customers the option of paying off their various bill payments. This would give us the opportunity to expand our technologies, making use of various web services. One of the examples of a web service we can use is with HSBC at <http://www.us.hsbc.com/internetbanking/billpay.html>.
- Another service we have discussed implementing in the future is an e-mail alert service. For example, when someone registers a new account or makes a transaction they may want confirmation of the action to be sent to their e-mail account. This could also be used as a mechanism to further validate someone to make sure that they are providing a real e-mail address.