

**ZappCash: An application Development Using C#  
Programming Language**



**A Capstone Project Documentation**

Presented to  
the College of Engineering and Architecture of  
Mapúa Malayan Colleges Mindanao, A Mapúa School  
Gen. Douglas MacArthur Highway, Davao City

In Partial Fulfillment  
of the Requirements for the Course of  
CPE142L OBJECT ORIENTED PROGRAMING  
LABORATORY

Proponents:  
Tomas, Vincent Alfred B. (A261)  
Zaspa, Anfernee Gerald M. (A261)

October 17, 2022



## **Introduction**

ZappCash is an accounting application that enables its users to monitor and control their finances. Inspired by the software GnuCash, it aims to manage the users' financial data for future reference. In addition, the project is a great alternative to Excel and the hassles that come along with managing and programming numerous cells and spreadsheets.

The idea for the application was birthed in the hope of providing individuals or groups the liberty to easily keep track of their finances, allowing them to develop smarter financial habits.

Showcase Video: <https://youtu.be/5T3eotyn0w0>

## Objectives

The objectives of this application include the following:

### *Create a double entry bookkeeping system*

The project will implement a method of recording such that transactions will reflect as debit or credit in at least two separate accounts.

### *Provide multiple account support*

The project will allow the creation and management of multiple financial accounts and their respective transactions.

### *Implement .zappcash file system*

The project will create files that the user may open to load up the accounts and transactions saved within it.

### *Design a timely and easy-to-use Graphical User Interface*

The project will apply an intuitive design to the application's user interface that will enable users to easily interact with the application and navigate through its components. The goal is also to set the application apart from GnuCash by improving on the latter's outdated appearance.

## Scope and Limitations

ZappCash is an accounting application. It only deals with the recording of transactions by the user and does not really have the capabilities to implement them. The ZappCash files created by the project are also inaccessible by other software. Furthermore, the project does not possess any current functionality that allows it to interact with other software and applications. As an example, ZappCash cannot export any of the saved financial data to programs like Microsoft Excel.

Some missing features for the application include but are not limited to:

- **Duplicate transaction function**

The project will be able to duplicate a certain transaction. The balance or amount for the accounts involved should change accordingly.

- **Duplicate account function**

The project will be able to duplicate a selected account. The balance or amount for the affected parent accounts should change accordingly.

- **Export transactions as CSV files**

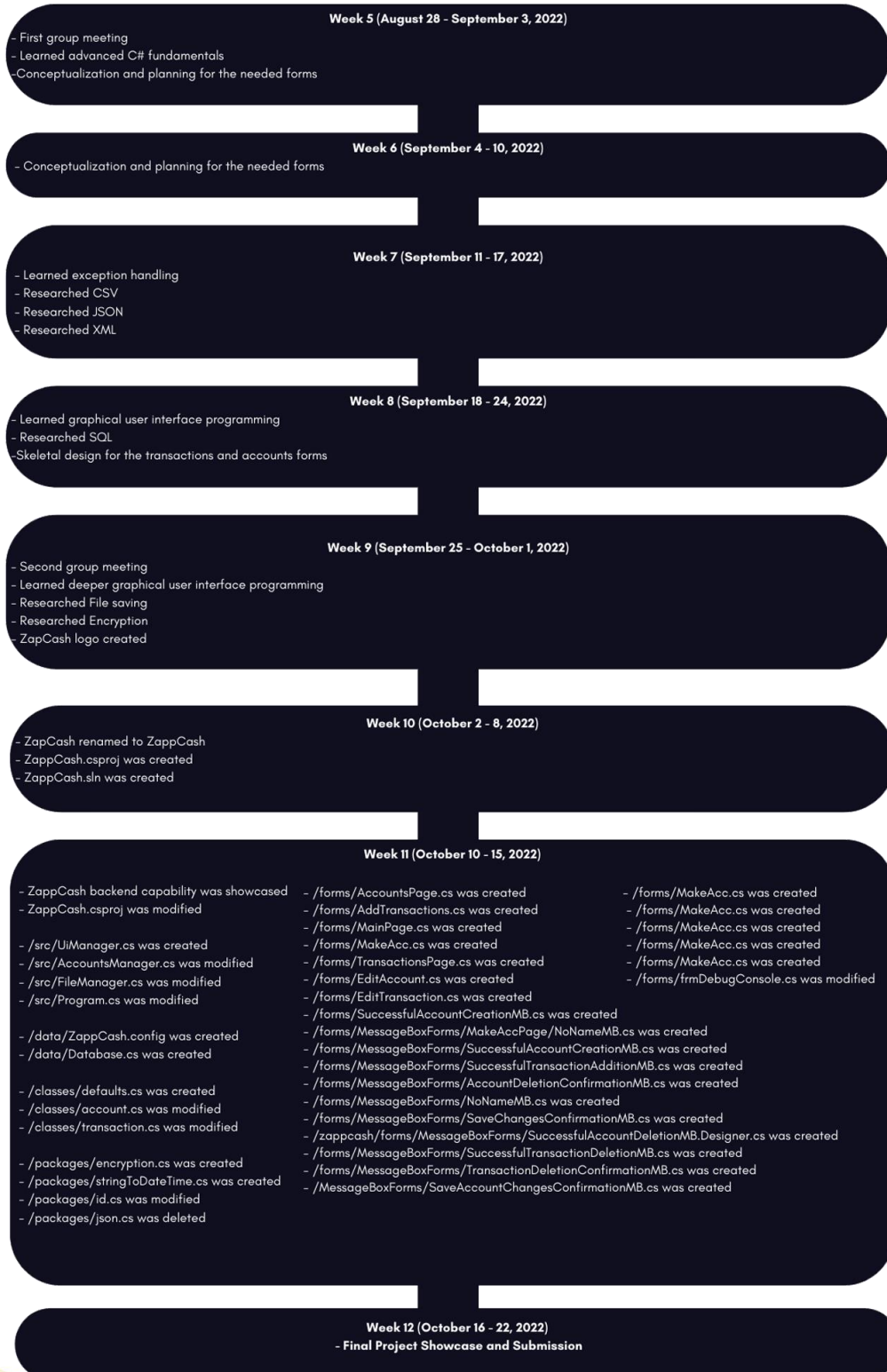
The project will be able to export ZappCash files as CSV files. This allows spreadsheet programs like Microsoft Excel and Google Spreadsheets to access the data within the files.

- **Export report function**

The project will be able to export a financial report of a given ZappCash file in PDF format.

## Discussion

The development of the project is as follows:

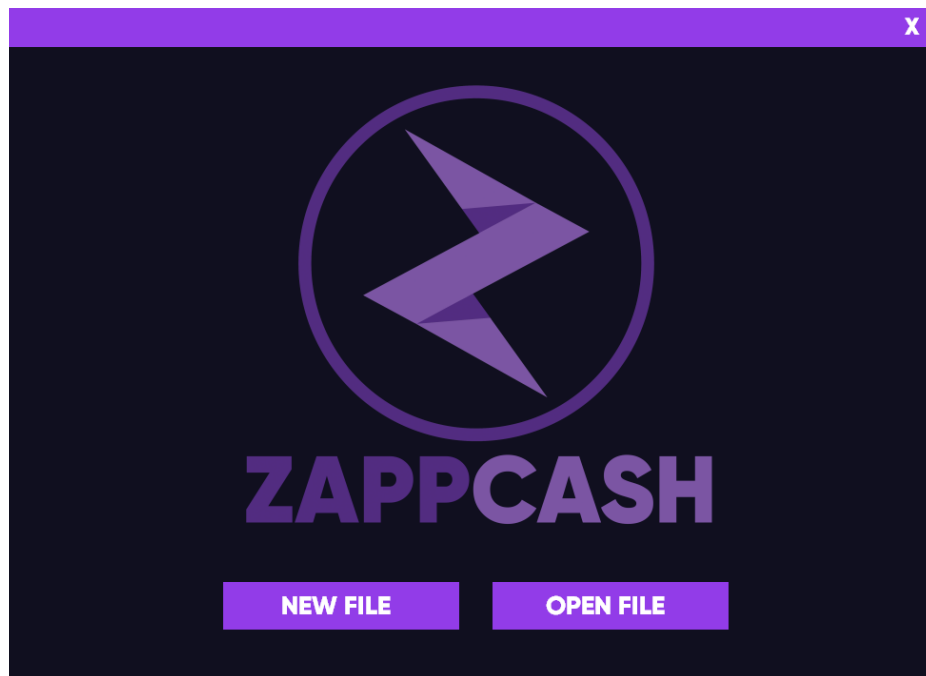




## Results

Below are the screenshots of the project output. These are all of the forms that are utilized and displayed by the application.

### Main Menu



*Figure 1. Main Menu Window (Not to scale).*

## Accounts Page

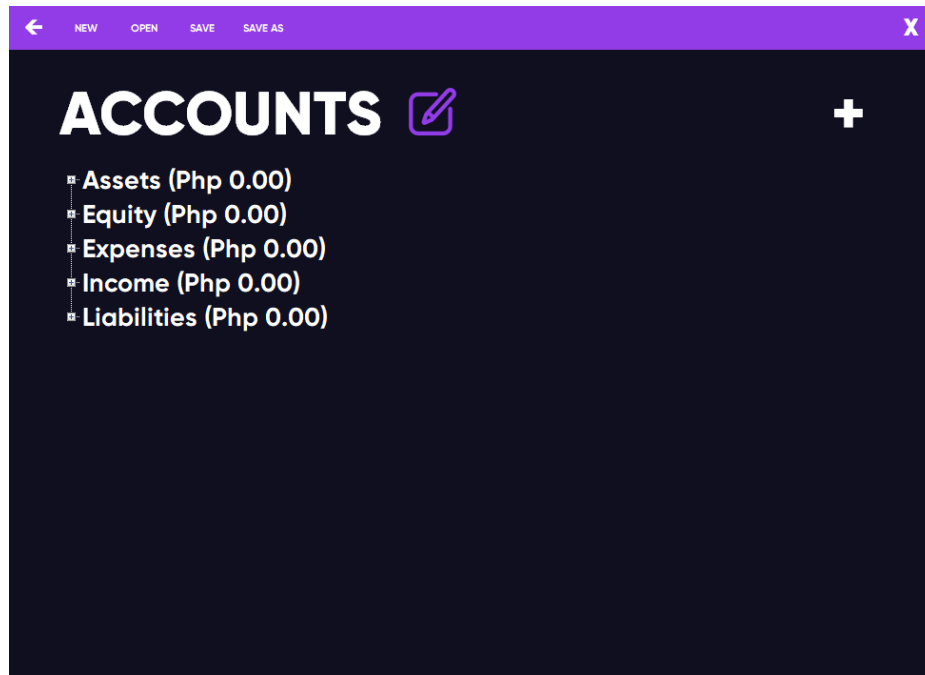


Figure 2. Accounts Page (Empty) (Not to scale).

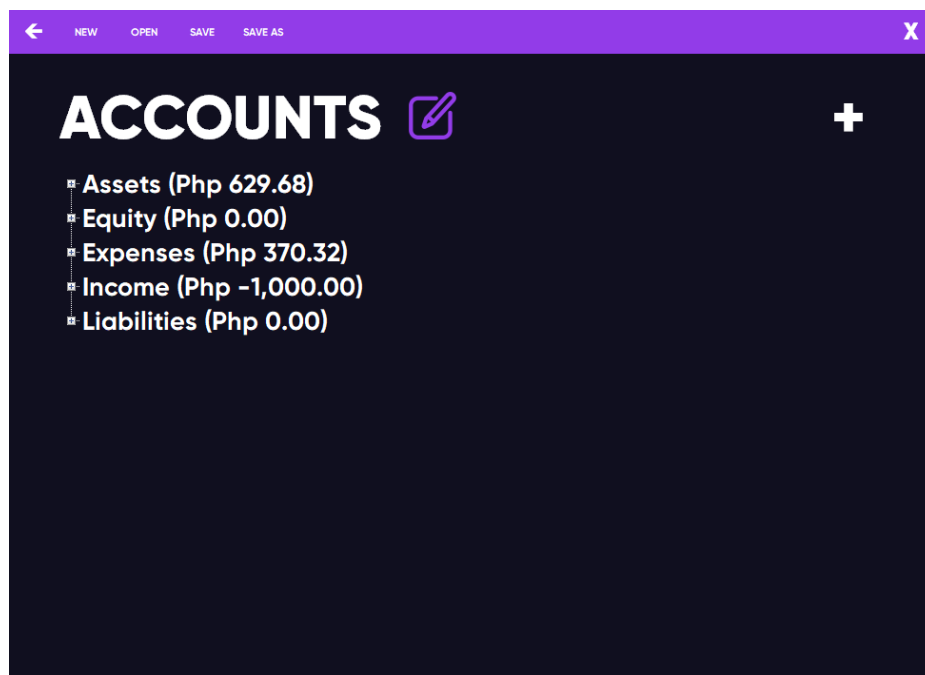


Figure 3. Accounts Page (Populated) (Not to scale).



## Transactions Page

Date	Num	Description	Transfer	Amount	Balance
------	-----	-------------	----------	--------	---------

Figure 4. Transactions Page (Empty) (Not to scale).

Date	Num	Description	Transfer	Amount	Balance
10/17/2022		Weekly Allowance	Allowance	Php 1,000.00	Php 1,000.00
10/17/2022		Transportation	Public Transportat...	Php (30.00)	Php 970.00
10/17/2022		Lunch	Dining	Php (124.00)	Php 846.00
10/17/2022		Deposit	Savings Account	Php (500.00)	Php 346.00
10/17/2022		Food	Groceries	Php (216.32)	Php 129.68

Figure 5. New File Page (Populated) (Not to scale).





## New Account Page

← X

**ACCOUNT NAME**

**PARENT ACCOUNT**

**PLACEHOLDER** ☐

**DESCRIPTION**

**OPENING BALANCE**

**CREATE ACCOUNT**

Figure 6. New Account Page.

## Edit Account Page

← X

**ACCOUNT NAME**

**PARENT ACCOUNT**

**PLACEHOLDER** ☒

**DESCRIPTION**

**SAVE CHANGES**

Figure 7. Edit Account Page.

## New Transaction Page

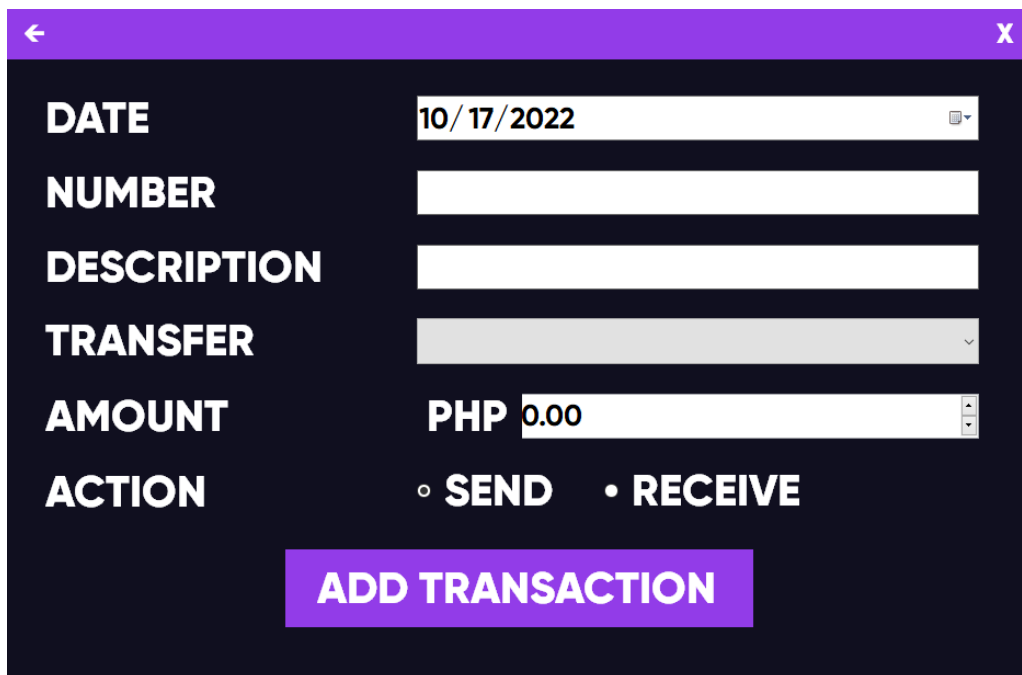
A screenshot of a mobile application interface for creating a new transaction. The background is dark blue. At the top is a purple header bar with a back arrow on the left and a close 'X' icon on the right. Below the header, there are several input fields: 'DATE' with a date picker showing '10/17/2022', 'NUMBER' with an empty text field, 'DESCRIPTION' with an empty text field, 'TRANSFER' with a dropdown menu showing a grey bar, 'AMOUNT' with a currency selector set to 'PHP' and a value of '0.00', and 'ACTION' with two radio buttons, 'SEND' (selected) and 'RECEIVE'. At the bottom center is a large purple button with the text 'ADD TRANSACTION' in white.

Figure 8. New Transaction Page.

## Edit Transaction Page

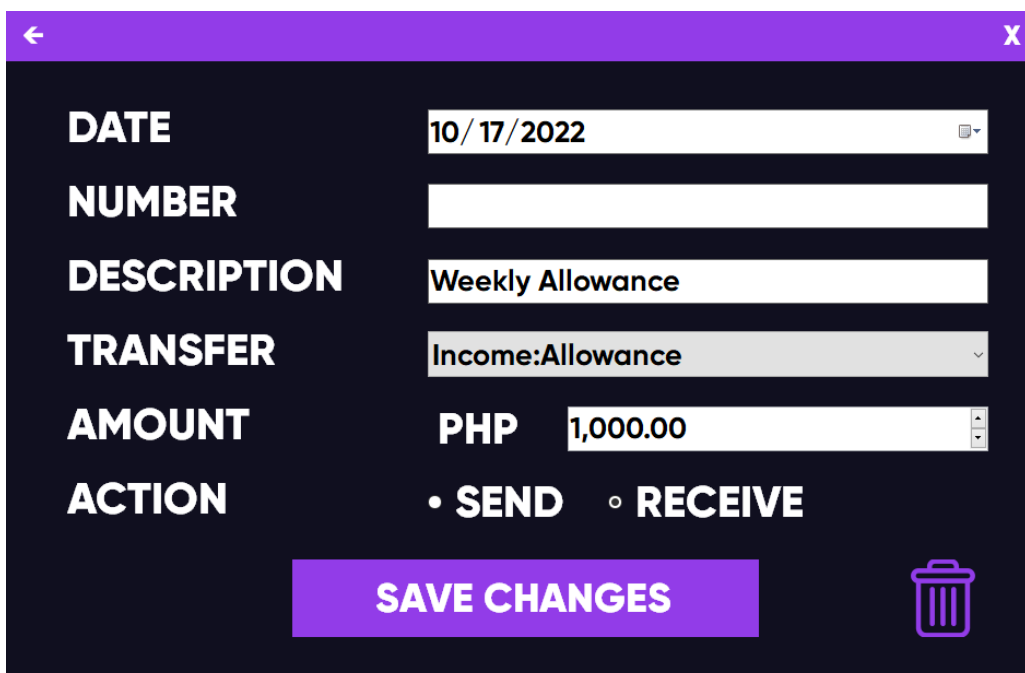
A screenshot of a mobile application interface for editing an existing transaction. The background is dark blue. At the top is a purple header bar with a back arrow on the left and a close 'X' icon on the right. Below the header, there are several input fields: 'DATE' with a date picker showing '10/17/2022', 'NUMBER' with an empty text field, 'DESCRIPTION' with a text field containing 'Weekly Allowance', 'TRANSFER' with a dropdown menu showing 'Income: Allowance', 'AMOUNT' with a currency selector set to 'PHP' and a value of '1,000.00', and 'ACTION' with two radio buttons, 'SEND' (selected) and 'RECEIVE'. At the bottom center is a large purple button with the text 'SAVE CHANGES' in white. To the right of this button is a purple trash can icon.

Figure 9. Edit Transaction Page.

## Prompts

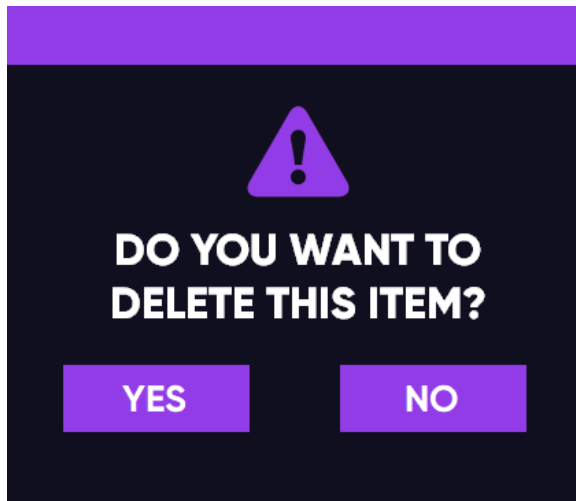


Figure 10. Delete Item Prompt (Not to scale).

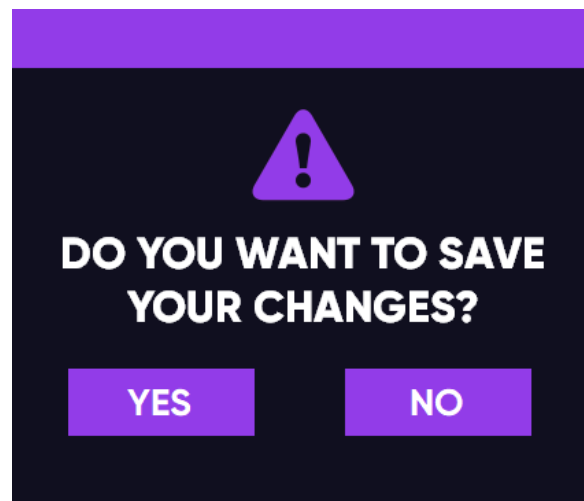


Figure 11. Save Changes Prompt (Not to scale).

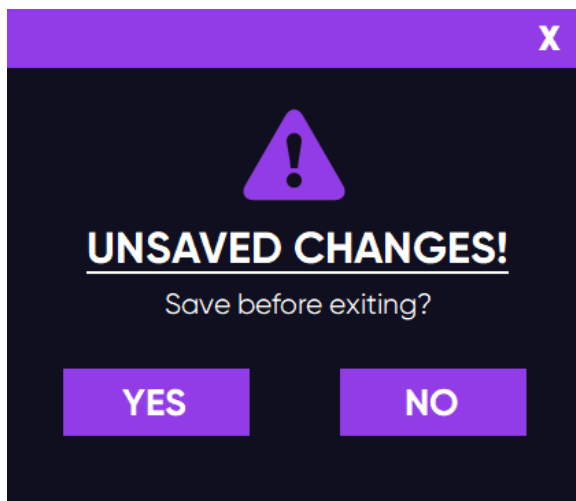


Figure 12. Unsaved Changes Prompt (Not to scale).

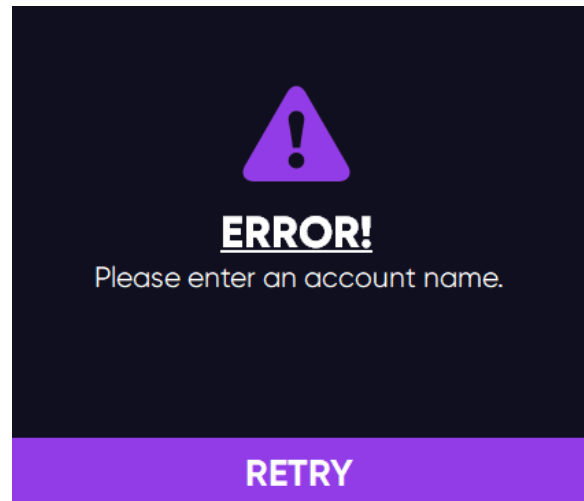


Figure 13. Error Prompt (Not to scale).



## Complete Code

GitHub Repository: <https://github.com/Kraftygames/ZappCash/>

### AccountsManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using ZappCash.classes;
using ZappCash.database;
using ZappCash.packages;

namespace ZappCash
{
    internal class AccountsManager
    {
        public static List<Account> GetAccounts()
        {
            return db_ZappCash.Accounts;
        }

        private static void setAccounts(List<Account> accounts)
        {
            db_ZappCash.Accounts = accounts;
            FileManager.TempSave();
        }

        public static void New()
        {
            db_ZappCash.CheckIntegrity();
            deleteAccounts();

            List<zC_DefaultAccountsProperty> defaultAccounts =
            db_ZappCash.Defaults.AccountDefaults.Accounts;

            foreach (zC_DefaultAccountsProperty defaultAccountProperties in
            defaultAccounts)
            {
                string parentId = GetAccountId(defaultAccountProperties.parent);
                NewAccount(Name: defaultAccountProperties.name, IsPlaceholder:
                defaultAccountProperties.isPlaceholder, ParentId: parentId);
            }
        }

        //General Actions
        public static void UpdateBalances()
        {
            List<Account> accounts = db_ZappCash.Accounts;
            accounts.Reverse();

            foreach (Account account in accounts)
            {
                long accountBalance = 0;
                foreach (Transaction transaction in account.Transactions)
                {
                    accountBalance += transaction.Amount;
                }
            }
        }
    }
}
```



```
        transaction.Balance = accountBalance;
    }

    foreach (Account child in GetAccountChildren(AccountId: account.Id))
    {
        accountBalance += child.Balance;
    }
    account.Balance = accountBalance;
}
accounts.Reverse();
setAccounts(accounts);
FileManager.TempSave();
}

private static void sortAccounts()
{
    List<Account> accounts = db_ZappCash.Accounts;
    accounts = accounts.OrderBy(account =>
GetLongAccountName(account.Id)).ToList();
    db_ZappCash.Accounts = accounts;
}

private static void sortTransations()
{
    List<Account> accounts = db_ZappCash.Accounts;
    foreach (Account account in accounts)
    {
        account.Transactions = account.Transactions.OrderBy(s =>
s.Date).ThenBy(s => s.Number).ToList();
    }
}

//Accounts
//get
public static Account GetAccount(string AccountId)
{
    List<Account> accounts = GetAccounts(); //import from database

    int accountIndex = GetAccountIndex(AccountId: AccountId);

    Account account;

    if (accountIndex < 0)
    {
        account = new Account(id:
db_ZappCash.Defaults.AccountDefaults.ParentId,
AccountAttributes("Root", "", "",
db_ZappCash.Defaults.AccountDefaults.AssetType,
db_ZappCash.Defaults.AccountDefaults.Decimals);
        attributes: new
"", ""), assetType:
isPlaceholder: true, null,
    }
    else
    {
        account = accounts[accountIndex];
    }

    return account;
}
public static int GetAccountIndex(string AccountId)
{
}
```





```
List<Account> accounts = GetAccounts();
int accountIndex = accounts.FindIndex(x => x.Id.Equals(AccountId));
return accountIndex;
}

public static List<Account> GetAccountChildren(string AccountId)
{
    List<Account> accounts = GetAccounts();
    List<Account> childAccounts = new List<Account>();
    foreach (Account account in accounts)
    {
        if (account.ParentId == AccountId)
        {
            childAccounts.Add(account);
        }
    }

    return childAccounts;
}

public static string GetAccountId(string accountName)
{
    List<Account> accounts = GetAccounts();
    foreach (Account account in accounts)
    {
        if (account.Attributes.Name == accountName)
        {
            return account.Id;
        }
    }
    return null;
}

public static string GetAccountId(string ParentAccountId, string AccountName)
{
    List<Account> accounts = GetAccountChildren(ParentAccountId);
    foreach (Account account in accounts)
    {
        if (account.Attributes.Name == AccountName)
        {
            return account.Id;
        }
    }
    return null;
}

//new
private static void NewAccount(Account Account)
{
    List<Account> accounts = GetAccounts();
    accounts.Add(Account);
    setAccounts(accounts);
    sortAccounts();
}
```

```
public static void NewAccount(string ParentId = null, byte Decimals = 2,
string Name = "", string Code = "", string Description = "", string Color = "#ffffff",
string Notes = "", string AssetType = "PHP", bool IsPlaceholder = false)
{
    AccountAttributes accountAttributes = new AccountAttributes(name: Name,
code: Code, description: Description, color: Color, notes: Notes);

    string accountId;

    if (ParentId == null)
    {
        string parentId = db_ZappCash.Defaults.AccountDefaults.ParentId;
        accountId = IdGenerator.GenerateID(parentId);
    }
    else
    {
        accountId = IdGenerator.GenerateID(ParentId);
    }

    Account account = new Account(id: accountId, attributes:
accountAttributes, assetType: AssetType, isPlaceholder: IsPlaceholder, parentId:
ParentId, decimals: Decimals);

    NewAccount(account);
}

//delete
private static void deleteAccount(string accountId) //raw delete an account
from database
{
    List<Account> accounts = GetAccounts(); //get from database
    int accountIndex = GetAccountIndex(accountId); //get account index
    accounts.RemoveAt(accountIndex); //delete account
    setAccounts(accounts); //write to database
    UpdateBalances();
}

private static void deleteAccounts()
{
    db_ZappCash.Accounts = new List<Account>();
} //delete all accounts

private static void deleteAccounts(string[] accountIds)
{
    foreach (string accountId in accountIds)
    {
        deleteAccount(accountId);
    }
} //delete selected accounts

public static void DeleteAccount(string AccountId, bool DeleteTransactions =
true, bool DeleteChildren = true)
{
    List<Account> accounts = GetAccounts(); //read from database
    List<string> selectedAccounts = new List<string>();

    if (DeleteChildren)
    {
        foreach (Account account in accounts)
        {
```



```
        if (account.ParentId == AccountId)
        {
            selectedAccounts.Add(account.Id);
        }
    }

    foreach (string accountId in selectedAccounts)
    {
        DeleteAccount(AccountId: accountId, DeleteTransactions = true,
DeleteChildren = true);
    }

}

if (DeleteTransactions)
{
    List<string> selectedtransactions = new List<string>();
    {
        foreach (Transaction transaction in GetTransactions(AccountId))
        {
            selectedtransactions.Add(transaction.Id);
        }
    }

    foreach (string transactionid in selectedtransactions)
    {
        DeleteTransaction(transactionid);
    }
}

deleteAccount (AccountId);
}

//edit
public static void EditAccount(string AccountId, string Name = null, string
Code = null, string Description = null, string Color = null, string Notes = null,
bool? IsPlaceholder = null, string ParentId = null, byte? Decimals = null)
{
    Account account = GetAccount(AccountId);
    int accountIndex = GetAccountIndex(AccountId);

    if (Name != null) { account.Attributes.Name = Name; }
    if (Code != null) { account.Attributes.Code = Code; }
    if (Description != null) { account.Attributes.Description = Description;
}

    if (Color != null) { account.Attributes.Color = Color; }
    if (Notes != null) { account.Attributes.Notes = Notes; }
    if (IsPlaceholder != null) { account.IsPlaceholder = (bool)IsPlaceholder;
}

    if (ParentId != null) { account.ParentId = ParentId; }
    if (ParentId == db_ZappCash.Defaults.AccountDefaults.ParentId) {
account.ParentId = null; }
    if (Decimals != null) { account.Decimals = (byte)Decimals; }

    DeleteAccount(AccountId, false, false);
    NewAccount(account);
    UpdateBalances();
}
```



```
//Transactions
public static Transaction GetTransaction(string AccountId, string
TransactionId)
{
    UpdateBalances();
    List<Account> accounts = GetAccounts(); //import from database

    int accountIndex = GetAccountIndex(AccountId: AccountId);
    int transactionIndex = getTransactionIndex(accountId: AccountId,
transactionId: TransactionId);

    Transaction transaction =
accounts[accountIndex].Transactions[transactionIndex];

    return transaction;
}

public static List<Transaction> GetTransactions(string AccountId)
{
    Account account = GetAccount(AccountId);
    List<Transaction> transactions = account.Transactions;
    return transactions;
}

private static int getTransactionIndex(string accountId, string
transactionId)
{
    List<Account> accounts = GetAccounts(); //import from database
    int accountIndex = GetAccountIndex(AccountId: accountId);

    Account account = accounts[accountIndex];

    int transactionIndex = account.Transactions.FindIndex(x =>
x.Id.Equals(transactionId));

    return transactionIndex;
}

private static void NewTransaction(string accountId, Transaction transaction)
{
    //if (GetAccount(accountId).IsPlaceholder)
    if (false)
    {
        throw new InvalidOperationException($"Account {accountId} is a
placeholder account and cannot contain any transactions.");
    } //no transactions in placeholders

    List<Account> accounts = GetAccounts(); //import from database
    int accountIndex = GetAccountIndex(accountId);

    accounts[accountIndex].Transactions.Add(transaction);

    setAccounts(accounts); //export to database
    sortTransactions();
    UpdateBalances();
} //raw new transaction
```



```
private static void NewTransaction(string AccountID, string TransactionID,
string TransferAccountID, DateTime Date, string Number = "", string Description =
"", long Amount = 0) //with transaction id
{
    Transaction transaction = new Transaction(Id: TransactionID, TransferId:
TransferAccountID, Date: Date, Number: Number, Description: Description, Amount:
Amount);
    NewTransaction(AccountID, transaction);

    Transaction mirrorTransaction = new Transaction(Id: TransactionID,
TransferId: AccountID, Date: Date, Number: Number, Description: Description, Amount:
-Amount);
    NewTransaction(TransferAccountID, mirrorTransaction);
}

public static void NewTransaction(string AccountID, string TransferAccountID,
DateTime? Date = null, string Number = "", string Description = "", long Amount = 0)
{
    DateTime date;
    if (Date == null) { date = DateTime.Today; }
    else { date = Convert.ToDateTime(Date); }

    string transactionId = IdGenerator.GenerateID(AccountID);
    NewTransaction(AccountID: AccountID, TransactionID: transactionId,
TransferAccountID: TransferAccountID, Date: date, Number: Number, Description:
Description, Amount: Amount);
} //no transaction id

public static void DeleteTransaction(string TransactionID)
{
    List<Account> accounts = GetAccounts(); //import from database

    List<int> accountIndices = new List<int>();
    List<int> transactionIndices = new List<int>();

    //index accounts with transactions and transactions
    foreach (Account account in accounts)
    {
        foreach (Transaction transaction in account.Transactions)
        {
            if (transaction.Id == TransactionID)
            {
                string accountId = account.Id;
                int accountIndex = GetAccountIndex(accountId);

                int transactionIndex = getTransactionIndex(accountId:
accountId, transactionId: TransactionID);

                transactionIndices.Add(transactionIndex);
                accountIndices.Add(accountIndex);
            }
        }
    }

    //nuke all indexed transactions
    for (int i = 0; i < accountIndices.Count; i++)
    {
        accounts[accountIndices[i]].Transactions.RemoveAt(transactionIndices[i]);
    }
}
```





```
    }

    db_ZappCash.Accounts = accounts; //export to database
    UpdateBalances();
}

public static void EditTransaction(string AccountId, string TransactionId,
DateTime? Date = null, string Number = null, string Description = null, string
TransferId = null, long? Amount = null)
{
    Transaction transaction = GetTransaction(AccountId: AccountId,
TransactionId: TransactionId);

    if (Date != null) { transaction.Date = (DateTime)Date; }
    if (Number != null) { transaction.Number = Number; }
    if (Description != null) { transaction.Description = Description; }
    if (TransferId != null) { transaction.TransferId = TransferId; }
    if (Amount != null) { transaction.Amount = (long)Amount; }

    DeleteTransaction(TransactionId);
    NewTransaction(AccountID: AccountId, TransactionID: TransactionId,
TransferAccountID: transaction.TransferId, Date: transaction.Date, Number:
transaction.Number, Description: transaction.Description, Amount:
transaction.Amount);

    sortTransations();
    UpdateBalances();
}

public static string GetLongAccountName(string Id)
{
    Account account = GetAccount(Id);
    string parentLongAccountName = "";

    if (account.ParentId != null)
    {
        parentLongAccountName = GetLongAccountName(account.ParentId);
        return $"{parentLongAccountName}:{account.Attributes.Name}";
    }

    return $"{account.Attributes.Name}";
}

}
}
```

#### **FileManager.cs**

```
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.IO;
using System.Windows.Forms;
using ZappCash.classes;
using ZappCash.database;
using ZappCash.packages.encryption;
```



```
namespace ZappCash
{
    static class FileManager
    {
        public static void OpenFile()
        {
            DatabaseIntegrityCheck();

            OpenFileDialog dlg = new OpenFileDialog();
            dlg.Filter = "ZappCash Files (*.zappcash)|*.zappcash";
            if (dlg.ShowDialog() == DialogResult.OK)
            {
                string filePath = dlg.FileName;
                fileRecord accessFile = new fileRecord(filePath);
                db_ZappCash.AccessFile = accessFile; //export to database

                CreateTempFile();
                Read();
            }
        }

        private static void CreateTempFile()
        {
            DatabaseIntegrityCheck();

            fileRecord accessFile = db_ZappCash.AccessFile; //import from database

            string fileName = $"{accessFile.Name}.tmp";
            string fileDirectory = accessFile.Directory;

            string filePath = Path.Combine(fileDirectory, fileName);

            fileRecord tempFile = new fileRecord(filePath);

            File.Copy(accessFile.Path, tempFile.Path, true);

            db_ZappCash.TempFile = tempFile; //export to database
        }

        public static void TempSave()
        {
            DatabaseIntegrityCheck();

            string tempFilePath = db_ZappCash.TempFile.Path;

            if (tempFilePath == null)
            {
                //SaveAs();
                return;
            }

            List<Account> accounts = db_ZappCash.Accounts;

            string jsonSerialized = JsonConvert.SerializeObject(accounts);
        }
    }
}
```



```
File.WriteAllText(path: tempFilePath, contents:
AdvancedEncryptionStandard.Encrypt(jsonSerialized));

}

public static void Save()
{
    DatabaseIntegrityCheck();

    string accessFilePath = db_ZappCash.AccessFile.Path; //import from
database

    if (accessFilePath == null)
    {
        SaveAs();
        return;
    }

    TempSave();

    string tempFilePath = db_ZappCash.TempFile.Path;
    File.Copy(tempFilePath, accessFilePath, true);
}

public static void SaveAs()
{
    DatabaseIntegrityCheck();

    SaveFileDialog dlg = new SaveFileDialog();
    dlg.Filter = "ZappCash Files (*.zappcash)|*.zappcash";
    dlg.DefaultExt = "zappcash";
    dlg.FileName = "Untitled";
    dlg.AddExtension = true;

    string tempFilePath = db_ZappCash.TempFile.Path;
    string accessFilePath = null;

    if (dlg.ShowDialog() == DialogResult.OK)
    {
        accessFilePath = dlg.FileName;
        db_ZappCash.AccessFile = new fileRecord(accessFilePath);
    }
    else { return; }

    if (db_ZappCash.TempFile.Path != null)
    {
        TempSave();
        File.Copy(tempFilePath, accessFilePath, true);

        return;
    }

    if (db_ZappCash.TempFile.Path == null)
    {
        List<Account> accounts = db_ZappCash.Accounts;
        string jsonSerialized = JsonConvert.SerializeObject(accounts);
        File.WriteAllText(path: accessFilePath, contents:
AdvancedEncryptionStandard.Encrypt(jsonSerialized));
        CreateTempFile();
    }
}
```

```

        return;
    }
}

private static void Read()
{
    string filePath = db_ZappCash.TempFile.Path; //import from database

    string fileContent = File.ReadAllText(filePath);

    List<Account> accounts =
JsonConvert.DeserializeObject<List<Account>>(AdvancedEncryptionStandard.Decrypt(fileContent));

    db_ZappCash.Accounts = accounts; //export to database
}

private static void DatabaseIntegrityCheck()
{
    db_ZappCash.CheckIntegrity();
}

public static void ReloadDefaults()
{
    string rootDirectory =
System.Reflection.Assembly.GetExecutingAssembly().Location;
    string rootPath = Path.GetDirectoryName(rootDirectory);
    string configPath = @"data\ZappCash.config";
    configPath = System.IO.Path.Combine(rootPath, configPath);

    zc_Defaults defaults =
JsonConvert.DeserializeObject<zc_Defaults>(File.ReadAllText(configPath));

    db_ZappCash.Defaults = defaults; //export to database
}

public static void TempDelete()
{
    db_ZappCash.CheckIntegrity();
    string tempPath = db_ZappCash.TempFile.Path;
    if (File.Exists(tempPath))
    {
        File.Delete(tempPath);
    }
}

public static void SaveBackup()
{
    //return;

    DatabaseIntegrityCheck();

    string tempFilePath = db_ZappCash.AccessFile.Path; //import from database
    string accessFilePath = db_ZappCash.AccessFile.Path;
    string accessFileExption = db_ZappCash.AccessFile.Extension;

    string autoSaveDate = $"{DateTime.Now.ToString("yyyyMMddHHmmss")}";

```



```
string autoSavePath =
$"{accessFilePath}.{autoSaveDate}{accessFileExpention}";

File.Copy(tempFilePath, autoSavePath, true);
}

public static void ResetDatabase()
{
    db_ZappCash.Reset();
}

public static bool IsAllSaved()
{
    if (db_ZappCash.AccessFile.Path == null || db_ZappCash.TempFile.Path ==
null)
    {
        return false;
    }

    if (File.Equals(db_ZappCash.AccessFile.Path, db_ZappCash.TempFile.Path))
    {
        return true;
    }

    SaveBackup();
    return false;
}

static bool FileEquals(string path1, string path2)
{
    byte[] file1 = File.ReadAllBytes(path1);
    byte[] file2 = File.ReadAllBytes(path2);
    if (file1.Length == file2.Length)
    {
        for (int i = 0; i < file1.Length; i++)
        {
            if (file1[i] != file2[i])
            {
                return false;
            }
        }
        return true;
    }
    return false;
}

}
```

#### **Program.cs**

```
using System;
using System.Windows.Forms;
using ZappCash.forms;

namespace ZappCash
{
    internal static class Program
    {
        /// <summary>
```





```
/// The main entry point for the application.  
/// </summary>  
[STAThread]  
static void Main()  
{  
    FileManager.ReloadDefaults();  
  
    Application.EnableVisualStyles();  
    Application.SetCompatibleTextRenderingDefault(false);  
  
    bool DebugMode = false;  
  
    if (!DebugMode)  
    {  
        Application.Run(new MainPage());  
    }  
    else  
    {  
        Application.Run(new frmDebugConsole());  
    }  
  
    FileManager.TempDelete();  
}  
}
```

## **Conclusion and Recommendation**

ZappCash was able to achieve all of the objectives set for the project. Certain improvements can be made to the application, starting with the addition of the lacking features stated under Scopes and Limitations. These are:

- Duplicate transaction function
- Duplicate account function
- Export transactions as CSV files
- Export report function

On top of this, below are some of the possible changes that would elevate the application even further. These additions are extremely encouraged for future developers that want to take on the project.

- Memory usage optimization
- Application hard disk size optimization
- Save file optimization
- Program architecture overhaul
- Database overhaul
- Encryption overhaul
- Release installer
- Save file icon
- Embed design font to application



## References

- Coding Droplets. (2021, March 29). *Serialize and Deserialize Json to C# [Step By Step Tutorial of JSON in C#]*. Retrieved from YouTube: <https://youtu.be/hLYHE1kIOpo>
- drf. (2014, June 15). *Convert Aes.Key to SecureString in C#*. Retrieved from Stack Overflow: <https://stackoverflow.com/questions/24226664/convert-aes-key-to-securestring-in-c-sharp>
- Ivanov, K. (2011, February 2). *How can I sort a List<T> by multiple T.attributes?* Retrieved from Stack Overflow: <https://stackoverflow.com/questions/4875737/how-can-i-sort-a-listt-by-multiple-t-attributes>
- Lamb, J. (2018, June 26). *C# DateTime to "YYYYMMDDHHMMSS" format*. (V. Ovchinnikov, Editor) Retrieved from Stack Overflow: <https://stackoverflow.com/questions/3025361/c-sharp-datetime-to-yyyyymmddhhmmss-format>
- Microsoft. (2022, September 16). *Aes Class*. Retrieved from Microsoft Learn: <https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.aes>
- Microsoft. (2022, Septmeber 30). *C# documentation*. Retrieved from Microsoft Learn: <https://learn.microsoft.com/en-us/dotnet/csharp/>
- Misja.com. (2022). *Epoch & Unix Timestamp Conversion Tools*. Retrieved from RpochConverter: <https://www.epochconverter.com/>
- RapidTables.com. (2022). *Decimal to Hexadecimal converter*. Retrieved from RapidTables: <https://www.rapidtables.com/convert/number/decimal-to-hex.html>
- Rigutini, R. (2022, May 30). *Common Practices In .NET Project Structure*. Retrieved from C#Corner: <https://www.c-sharpcorner.com/article/common-practices-in-net-project-structure/>
- rodbv. (2009, January 4). *generate treeview nodes programmatically*. Retrieved from Stack Overflow: <https://stackoverflow.com/questions/410978/generate-treeview-nodes-programmatically>
- Seibar. (2008, September 19). *.NET String.Format() to add commas in thousands place for a number*. Retrieved from Stack Overflow: <https://stackoverflow.com/questions/105770/net-string-format-to-add-commas-in-thousands-place-for-a-number>
- SuncoastSoftware. (2013, February 24). *C# Global Variable Tutorial*. Retrieved from YouTube: <https://youtu.be/A4Zv6qDIvi0>



**MAPUA**  
MALAYAN COLLEGES  
MINDANAO

ZappCash: An application Development Using C#

A Capstone Project Documentation

*Reserved for Future Use*