

# Final project documentation



**SNOW  
COLLEGE**

Christopher Hardamek

CS1410

Due Date 04.23.22

## Table of contents

<b>Introduction.....</b>	<b>3</b>
<b>Tasks.....</b>	<b>3</b>
<i>Task 1: Program does something fun or useful .....</i>	<i>3</i>
<i>Task 2: Program handles invalid input.....</i>	<i>3</i>
<i>Task 3: Logic and IO cleanly separated.....</i>	<i>5</i>
<i>Task 4: UI layer calls down into logic layer for processing .....</i>	<i>5</i>
<i>Task 5: Read and/or write data from one or more: file, database, network.....</i>	<i>6</i>
<i>Task 6: Use interfaces to define access to external services (e.g., your storage service, database, network, etc.) .....</i>	<i>6</i>
<i>Task 7: Use bogus/test implementations of those interfaces in your unit tests (e.g., do _not_ actually write to the filesystem or call the network from a unit test). .....</i>	<i>6</i>
<i>Task 8: Create a real implementation of your service interface(s) that actually writes to disk / communicates on the network in your real application.....</i>	<i>8</i>
<b>Retrospective.....</b>	<b>10</b>
<i>Retrospective: What you learned in the process of completing this project .....</i>	<i>10</i>
<i>Retrospective: Share which principles/patterns/technologies you enjoyed learning about and why you feel they're valuable/beneficial. ....</i>	<i>10</i>
<i>Retrospective: Discuss things you'll do differently next project. ....</i>	<i>11</i>

## Introduction

This documentary is about my final project in CS1410 Object-oriented Programming. The target of this final project is to know how to program and how to handle different situations. Here is my [Repository](#).

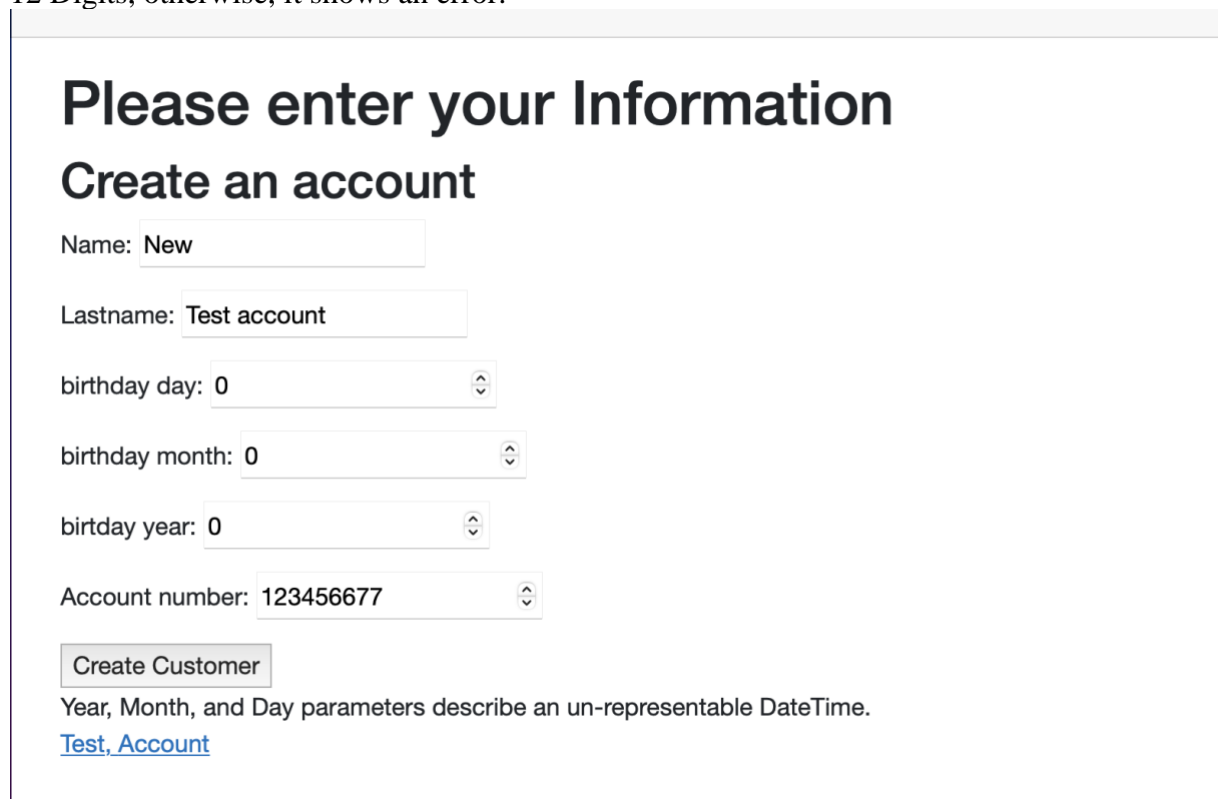
## Tasks

### Task 1: Program does something fun or useful

This program can create accounts. The user can create a saving account, make a deposit, withdraw, and get credit from the bank in the accounts. Has a own Account number.

### Task 2: Program handles invalid input

On the Index page, there are some input fields. One of the Input fields is Name, it isn't possible to create an account with just numbers. This counts for Last name and name. The other inputs are for the day, month, and year. It isn't possible to create a name as a date. And this program will check if the date exists. The Account number must be between 8 and 12 Digits, otherwise, it shows an error.



**Please enter your Information**

**Create an account**

Name:

Lastname:

birthday day:

birthday month:

birthday year:

Account number:

Year, Month, and Day parameters describe an un-representable DateTime.

[Test, Account](#)

Screenshot 1

On screenshot 1, you can see there is an Error message because the date is not valid. The same happens with the name and Lastname. The requirement is not empty and minimum of 2 Letters. On screenshot 2 there is an error message which saw invalid name on name. On screenshot 3 there is the error the name is short.

# Please enter your Information

## Create an account

Name:

Lastname: Hardamek

birthday day: 1

birthday month: 1

birthday year: 1

Account number: 123455678

Create Customer

Object reference not set to an instance of an object.

Screenshot 2

# Please enter your Information

## Create an account

Name: Test

Lastname: a

birthday day: 1

birthday month: 1

birthday year: 1

Account number: 12345678

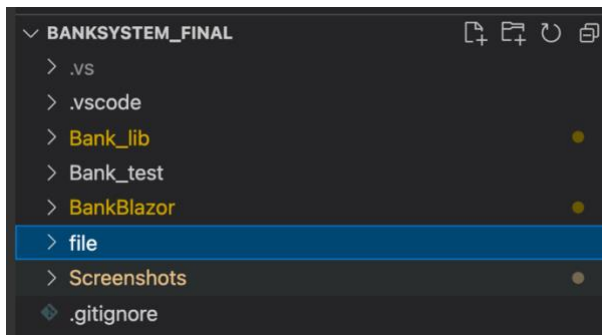
Create Customer

Account Name must be minimum 2 letters

Screenshot 3

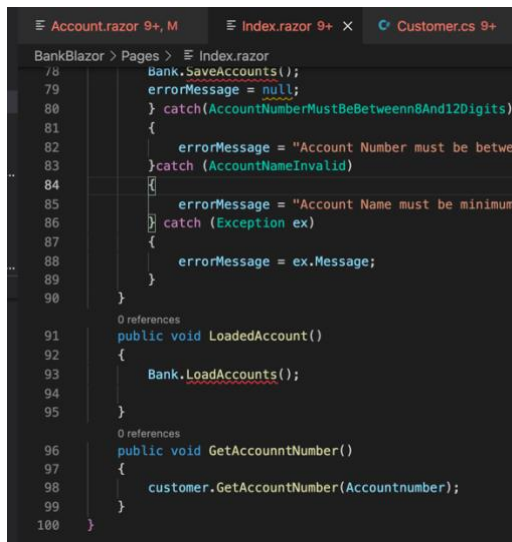
### Task 3: Logic and IO cleanly separated

On Screenshot 4 , there is 4 Folder. One folder Bank\_lib. In this folder is the whole program



Screenshot 4

code. The folder BankBlazor, in this folder, is the Web application. The last folder is Bank\_test. There is all the test inside. The last folder is for the files. In this folder there is the accounts.json and logs.json inside. In the Finalproject.sln there is the reference between the folders. It is necessary to have this because without the reference is not possible to use all folders and the logic.



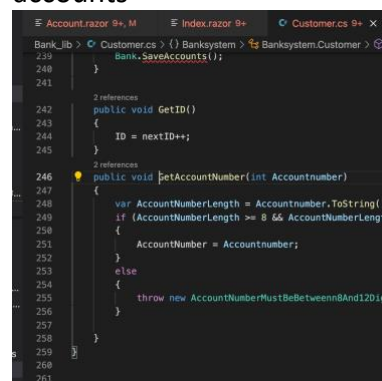
Screenshot 5

### Task 4: UI layer calls down into logic layer for processing

In screenshot 5 there is the index page. On line 96 is called the function

GetAccountNumber(Accountnumber). In screenshot 4 there is the function of screenshot 3

There is another function called a function in the other folder. On Line 92 public void LoadAccount(). There is a method called Bank.LoadAccounts(). This a function on in the Bank.cs which is called load accounts



Screenshot 6

## Task 5: Read and/or write data from one or more: file, database, network

```
{ } accounts.json M x
{ } accounts.json > { } 1 > { } Owner
1  [
2    {
3      "Owner": {
4        "Name": "Test",
5        "Lastname": "Account",
6        "Age": 0,
7        "Birthdate": "0001-01-01T00:00:00",
8        "CheckingBalance": 50,
9        "SavingAccount": false,
10       "Credit": false,
11       "SavingBalance": 0,
12       "log": null,
13       "CreditAmount": 0,
14       "ID": 1,
15       "AccountNumber": 12345678
16     }
17   },
18   {
19     "Owner": {
20       "Name": "Try",
21       "Lastname": "Two",
22       "Age": 0,
23       "Birthdate": "0001-01-01T00:00:00",
24       "CheckingBalance": 50,
25       "SavingAccount": false,
26       "Credit": false,
27       "SavingBalance": 0,
28       "log": null,
29       "CreditAmount": 0,
30       "ID": 2,
31       "AccountNumber": 12345678
32     }
33   }
34 ]
```

Screenshot 7

In my project, I work with 2 files. The first file is responsible for the and the other one is responsible for the logs.

In screenshot 5 there are two accounts. One of them is Test Account and the other one is Try Two. They both have a different customerID. Test Account has ID 1 and "Try two" has ID 2. This number is unique.

Test Account has a balance of \$50 and the account number is 123455678. The Try Two account has a balance of \$50 and the account number is 12345678.

On the other hand, there is more information about the accounts. For example, they were both born on the 1<sup>st</sup> of January in the year 0001. Both accounts don't have a Credit. The saving balance has a balance of \$0 in both accounts. The Age function is not yet implemented.

There is another file what you can see on screenshot 8

## Task 6: Use interfaces to define access to external services (e.g., your storage service, database, network, etc.)

I don't get it, because I got your help, but it didn't do anything.

## Task 7: Use bogus/test implementations of those interfaces in your unit tests (e.g., do not actually write to the filesystem or call the network from a unit test).

```
0 references | Run | Test | Debug | Test
33 | public void CustomerIsNotSeniorCitizen()
34 | {
35 |     var c = new Customer("Peter", "Lastname", 12, 12, 2000, 5000, false, 0, false, 0, 12345678);
36 |     Assert.IsFalse(c.IsSenior);
37 | }
```

Screenshot 8

In screenshot 8 it will check if the Customer is over 45. It's failing.

```
62 |
63 | 1 reference
63 | public bool IsSenior => Birthdate < DateTime.Today.AddYears(-45);
64 |
65 |
```

Screenshot 9

Screenshot 9 shows that the test is passing. It takes the Birthdate and takes it minus 45, and when it's over 0 it will pass.

```
39 | [Test]
    | 0 references | Run Test | Debug Test
40 | public void IsCustomerYoungerTheFuture()
41 | {
42 |     var c = new Customer("Peter", "Lastname", 12, 12, 2000, 5000, false, 0, false, 0, 12345678);
43 |     Assert.IsFalse(c.IsYounger);
44 | }
```

Screenshot 10

In screenshot 10 there is one test that proves the customer is younger than the future. In this case, it will fail because the test doesn't exist.

```
64 | 1 reference
    | public bool IsYounger => Birthdate < DateTime.Now;
65 |
```

Screenshot 11

In Screenshot 11 there is the implementation to check is someone born in the future.

```
45 | [Test]
    | 0 references | Run Test | Debug Test
46 | public void GetRightAccountnumber()
47 | {
48 |     (local variable) Customer customer
49 |     Customer customer = new Customer("Peter", "Lastname", 12, 12, 2000, 5000, false, 0, false, 0, 12345678);
50 |     Assert.AreEqual(12345678, customer.AccountNumber);
    | }
```

Screenshot 12

In screenshot 12 there is a test that gets the right Account Number. This function below is how to get the right Account number.

```
247 | 2 references
    | public void GetAccountNumber(int Accountnumber)
248 | {
249 |     var AccountNumberLength = Accountnumber.ToString().Length;
250 |     if (AccountNumberLength >= 8 && AccountNumberLength <= 12)
251 |     {
252 |         AccountNumber = Accountnumber;
253 |     }
254 |     else
255 |     {
256 |         throw new AccountNumberMustBeBetween8And12Digits();
257 |     }
258 |
259 | }
```

Screenshot 13

There is one check, the account number must be between 8 and 12 Digits. When not it will throw an exception

```

51 [Test]
    0 references | Run Test | Debug Test
52 public void getID()
53 {
54     Customer c1 = new Customer("PEter", "Lastname", 12, 12, 2000, 5000, false, 0, false, 0, 12345678);
55     Customer c2 = new Customer("Michael", "Sure", 12, 12, 2000, 5000, false, 0, false, 0, 12345678);
56     Assert.AreNotEqual(true, c1.ID == c2.ID);
57
58 }
    0 references

```

Screenshot 14

In this Test case, there will be a test is the ID auto-increment when I create multiple accounts.

Here is the solution that gets its automatic increment.

```

243 2 references
243 public void GetID()
244 {
245     ID = nextID++;
246 }

```

Screenshot 15

nextID this value is a static variable, and ID is an integer value. The value starts by counting from one.

```

59 public void CreateSavingAccount()
60 {
61     Customer c1 = new Customer("PEter", "Lastname", 12, 12, 2000, 5000, false, 0, false, 0, 12345678);
62     c1.MakeSavingAccount();
63     Assert.IsTrue(true);
64 }
65
66

```

Screenshot 16

This Test cases test the activating of a Saving account, the default value is false. With lone 62 it will activate, and in line 63 there, it will test that the value has the bool value true.

```

156 2 references
156 public void MakeSavingAccount()
157 {
158     foreach (var customer in Bank.Accounts)
159     {
160         if (customer.Owner.Name == Name)
161         {
162             customer.Owner.savingaccount = true;
163             Bank.SaveAccounts();
164             logging.logs.Add($"The user with the ID: {ID} savingaccount created for {Name} on {DateTime.Now}");
165             logging.SaveLog();
166             break;
167         }
168     }
169     Bank.SaveAccounts();

```

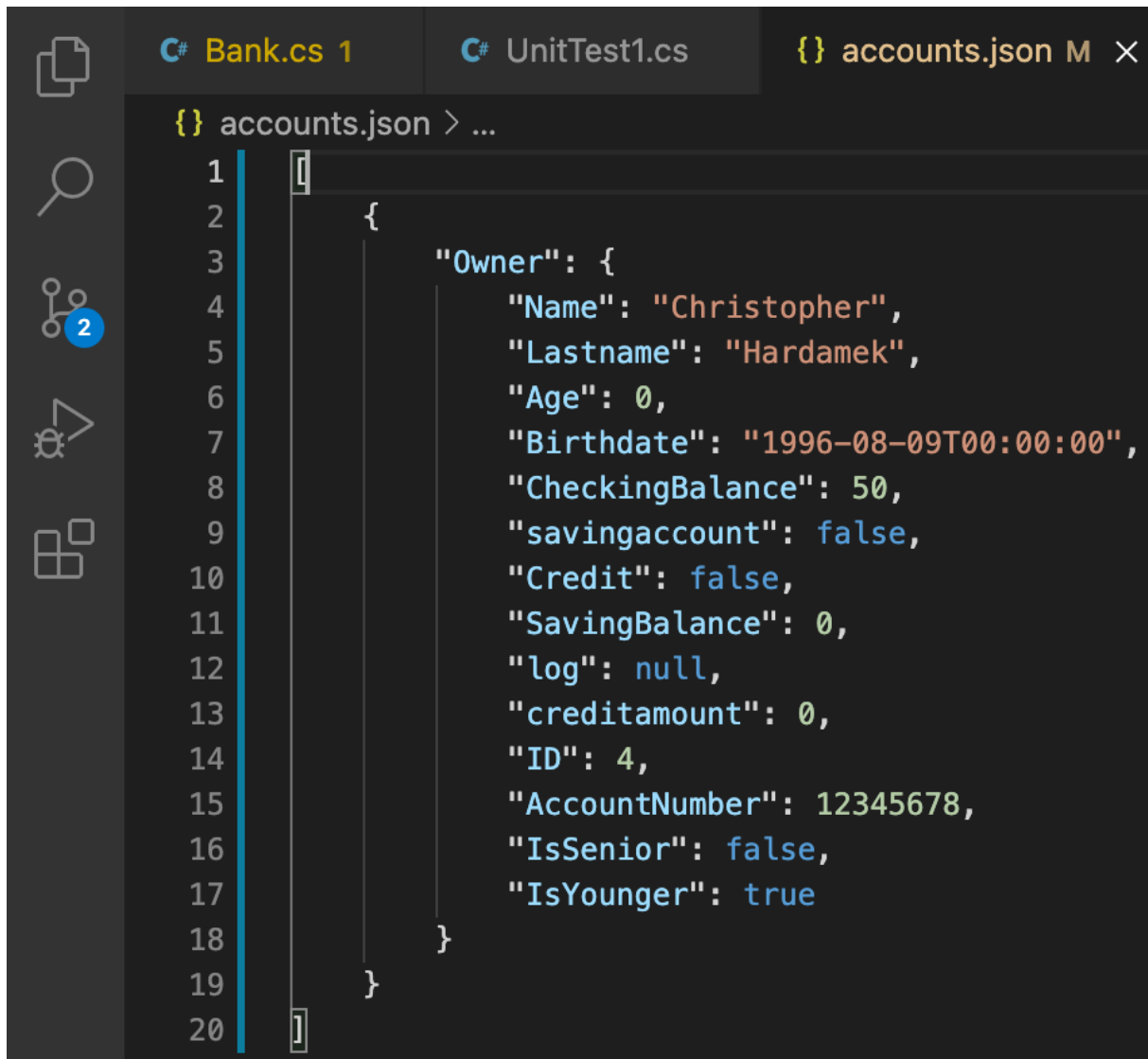
Screenshot 17

This function in screenshot 17 will activate the Value of the accounts. In line 158 it will scan all accounts in the list of Accounts. In the IF statement there will be checked, is there someone with the same name as the account, then will it activate it.

Task 8: Create a real implementation of your service interface(s) that actually writes to disk / communicates on the network in your real application.

In this task, It will create an account with my real name. In the JSON looks like this. See screenshot 18





```
{  
  "Owner": {  
    "Name": "Christopher",  
    "Lastname": "Hardamek",  
    "Age": 0,  
    "Birthdate": "1996-08-09T00:00:00",  
    "CheckingBalance": 50,  
    "savingaccount": false,  
    "Credit": false,  
    "SavingBalance": 0,  
    "log": null,  
    "creditamount": 0,  
    "ID": 4,  
    "AccountNumber": 12345678,  
    "IsSenior": false,  
    "IsYounger": true  
  }  
}
```

Screenshot 18

The Values of this account is right below on screenshot 19

## Please enter your Information

### Create an account

Name:

Lastname:

birthday day:

birthday month:

birthday year:

Account number:

Screenshot 19

I get all account information on the account page. On Screenshot 20 there is some

## Welcome Christopher, Hardamek

Current Checkingbalance: \$50.00

Deposit Amount 0

Make Deposit

withdrawn

### Account Information

Name: Christopher

Lastname: Hardamek

Balance: 50

Birhtdate:

Day: 9

Month: 8

Year: 1996

Savingaccount: False

Creditamount: 0

Account Number: 12345678

Create Savingaccount

Get Credit

0

### Activities:

when the user doesn't have the money to payback it will hide the payback button. All activities will be logged. For example when someone payin in the ID then will showed below under the "Activities". Here is an example. There is the ID and the time.

### Activities:

The user with the ID: 4 savingaccount created for Christopher on 4/23/2022 11:46:19 PM

The user with the ID: 4 adding money 2000 to Christopher new Balance: 2050 on 4/23/2022 11:46:30 PM

The user with the ID: 4 adding money 2000 to Christopher new Balance: 4050 on 4/23/2022 11:46:30 PM

The user with the ID: 4 adding money 2000 to Christopher new Balance: 6050 on 4/23/2022 11:46:30 PM

The user with the ID: 4 adding money 2000 to Christopher new Balance: 8050 on 4/23/2022 11:46:30 PM

The user with the ID: 4 adding money 2000 to Christopher new Balance: 10050 on 4/23/2022 11:46:30 PM

*Screenshot 20*

## Retrospective

[Retrospective: What you learned in the process of completing this project](#)

I learned much about programming. I learned how to work with files, how to work with exceptions, Blazor, debugging, Query expression, and Test-driven development. During I did my final project, it was sometimes hard because I stuck in problems that I had never.

I think that is normal for a final project respectively. What I learned too, don't give up, because success after the project is bigger.

[Retrospective: Share which principles/patterns/technologies you enjoyed learning about and why you feel they're valuable/beneficial.](#)

I enjoyed it. I think some of the functions I will need in the future. I enjoyed the time working with JSON. In this case, I learned the format of JSON. In the IT world, there are often files in JSON. I think JSON is important for config files are in JSON format and that is the first step of the introduction to JSON. In this project I did, there is a lot of option that can be improved, for example. With a log-in system, or transfer money between customers. But the basic is done. What else can be improved is the data get saved in the database. In most cases is the

pattern the same. I was introduced with this semester in c# that I hadn't before. I just learned c++.

Retrospective: Discuss things you'll do differently next project.

In the next project, I will inform more about which options I have to implement some functions instead of using just the basics. During this time, I will set some milestones. A mile is for me, I will set a milestone for example, until the weekend I want to have a log-in system, and the next weekend I want to have a User Interface. It will take more time, but this project took already a lot of time.