Chilindo Interview Technical Challenge

# Table of Contents

[Table of Contents 2](#_Toc482004316)

[High Level Overview 3](#_Toc482004317)

[Background 4](#_Toc482004318)

[Requirements 5](#_Toc482004319)

[Notes 5](#_Toc482004320)

[Web Services 6](#_Toc482004321)

# High Level Overview

As part of the Chilindo interview process, you have been tasked with transforming a set of requirements in to a proof-of-concept as part of a technical exercise. This will helps us determine how you approach problems and the way in which you build solutions.

There is no inherent time limit applied to the task and you will be expected to present your solution during the interview.

If you come across something which doesn’t make sense, either because some information is missing or you don’t agree with the implementation, **make a decision** on what should be done.

# Background

Chilindo is trialling an automated banking feature to evaluate whether it can be better handled by building internally. The first step is to implement a simple transaction processing framework. The framework will receive input via a Web Api and process those requests based on some simple rules. This should be backed by MSSQL.

The deliverables for the task are:

* A .zip file containing your source code (with all /bin/ and /obj/ folders removed).
* Sent via email the link
* A short presentation, in person, of your solution, telling us:
  + Overview of your implementation.
  + How far you got.
  + Biggest challenges.
  + Chosen libraries / frameworks

## Requirements

* Create a WebApi that supports three methods ‘balance’, ‘deposit’ and ‘withdraw’.
* The deposit method will accept requests which increase the amount of available funds for the associated bank account.
* The withdraw method will accept requests which deplete the amount of available funds for the associated bank account.
* A withdraw request should be declined if there are insufficient funds in the account.
* The solution should allow deposit & withdrawal for multiple currencies.
* The solution has to work across multiple instances for scalability.
* Create a simple client (console application or website) which acts as a request handler for the service.
* The client should also contain a feature that allows the service to be requested in an automatic and random fashion with a combination of withdraws and deposits taking place over a given period of time. Use this to show that the implementation handles concurrency well.

### Notes

In the test example, you can generate a small set of bank account numbers for use in your demo.

Authentication is not a concern in this implementation given that it’s an internal proof-of-concept, assume that all servers involved will be running over internal, closed networks.

The implementation must be secure in as much as only the amount of funds available in the account can be withdrawn. I.e. the account can never have a negative balance. This will require a strong implementation to deal with many concurrent users.

The POC must prove that the framework works and will scale to large volumes of concurrent users. This will be theoretical as it’s unlikely we’ll run this in a load test scenario, however you need to explain how your solution handles this.

A simple client should also be created to interact with the services and to demo the concurrency capabilities of the services. This can be carried out using either a simple console application or a website.

Include as many unit tests as possible. (Preferably NUnit / MS test framework)

Include Transaction Logging for Deposits & Withdrawals

Include Exception Handling

When presenting the solution, be prepared to discuss ways in which we could secure the services.

## WebApi EndPoints :

The below endpoints are required for this test.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Method** | Balance | **HTTP Type** | | GET |
| An endpoint used to retrieve account balance.  Example URL: <http://localhost/api/account/balance> | | | | |
| Parameter Name | Data Type | Required | Comment | |
| AccountNumber | Integer | Yes | The account number of which to retrieve the balance. | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Method** | Deposit | **HTTP Type** | | POST |
| An endpoint to deposit funds in an account.  Example URL: <http://localhost/api/account/deposit> | | | | |
| Parameter Name | Data Type | Required | Comment | |
| AccountNumber | Integer | Yes | The account number to deposit the funds to. | |
| Amount | Decimal | Yes | The value of funds being deposited. | |
| Currency | String | Yes | The currency of the funds being deposited. | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Method** | Withdraw | **HTTP Type** | | POST |
| A n endpoint used to withdraw funds from an account.  Example URL: http://localhost/account/withdraw | | | | |
| Parameter Name | Data Type | Required | Comment | |
| AccountNumber | Integer | Yes | The account number to deposit the funds to. | |
| Amount | Decimal | Yes | The value of funds being deposited. | |
| Currency | String | Yes | The currency of the funds being deposited. | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Service Response** | Response Parameters |  | |
| The result of a call to any of the account methods | | | |
| Parameter Name | Data Type | Required | Comment |
| AccountNumber | Integer | Yes | An identifier used to uniquely identify a single account. |
| Successful | bool | Yes | If true, the request succeeded and the required action was performed, otherwise false. |
| Balance | Decimal | No | The current available balance after the action was performed. Can be omitted or null if Successful=false. |
| Currency | String | No | The currency of the current available balance after the deposit was made, Can be omitted or null if Successful=false. |
| Message | String | Yes | A message detailing the outcome of the operation. If the request failed, this should detail why. |

# Solution for Technical Challenge

## Prerequisites

1. Visual Studio 2015 Enterprise V14.0.24720.00 or above
2. Microsoft SQL Server Management Studio 2008 V10.0.1600.22 or above
3. Google Chrome (for UI)

## Setup the Project

1. Restore DB backup from ‘DB/Backup/AccountsDB\_GEO.bak’.
2. If the restore fails, execute ' DB/ Scripts/Build Scripts.sql' for creating a db named 'ACCOUNTSDB\_GEO' in SQL.

*Note: This contains the scripts to drop the db if exists and then create a new db, create tables 'accountDetails', 'currencyDetails' and 'transactionLog'. The query also performs insert of dummy accounts with a pre-set balance. It also inserts some currencies with their conversion rate.*

1. The project solution is ‘AccountsAPP/AccountsAPP.sln’
2. Update the connection string 'ACCOUNTSDB\_GEOConnectionString' in web.config to perform DB operations.

## Details of on proposed solution

1. All the amount related operation is done by setting US Dollar as the base currency.
2. Even though there are options to perform a deposit/withdrawal in different currency, the code is done in such a way to convert the amount to US Dollar based on a predefined conversion rate (with reference to 'currencyDetails' table). Online live conversion rate is not considered to avoid an external dependency.
3. In addition to the required API list, a few more is added in the implementation, which is detailed in the API details.
4. The solution also provides an option to view a history of transactions done against a selected account.

### Solution for Concurrency

1. The ‘Update Check’ property of ‘accountBalance’ column of 'accountDetails' table data model is set to ‘Always’. Hence, any update to ‘accountBalance’ before the operation would raise a ‘ChangeConflictException’ in LINQ to SQL.
2. Upon the change conflict exception, the code is done to perform a recursive update of ‘accountBalance’ until to such time the change conflict exception is not occurred. Or ‘accountBalance’ has a probability to go below 0. Upon which the transaction will fail and the user will be notified.
3. Since ‘accountBalance’ column is the only entity that gets updated, ‘Update Check’ property is applied on ‘accountBalance’ column only. If there comes a situation when multiple columns has to be updated, it is better to use a ROWVERSION/TIMESTAMP column and perform ‘Update Check’ on it.

### Features in UI

1. Upon the launch of the tool, the user is expected to select an Account from the list that is provided.
2. Upon the selection of an account, the Current Balance and Last Modified date of the account will be displayed. In addition to that, there will be options to perform the following operations
   1. Add/Withdraw Amount
   2. Multiple Add/Withdraw Amount
   3. View Transaction History
3. Add/Withdraw Amount option expects the user to provide the following inputs.
   1. Transaction Type : Deposit/Withdraw
   2. Currency Type : from provided list
   3. Amount : transaction amount
4. Multiple Add/Withdraw Amount option expects the user to add as many transactions using ‘Add Transaction’. The user will be provided with options to input the same details that are required to perform an Add/Withdraw Amount option. Upon the Bulk Update, the code will loop through the list of transactions and perform the update using multiple API calls.
5. View Transaction History will load all the transactions done against the selected account.

### Progress of Implementation

1. The implementation is complete as per the stated requirements. With the following additions to the requirements.
   1. A UI to see a list of accounts based on 'accountDetails' table. The user is forced to select one of the account holder’ to perform any of the account related activities.
   2. A UI to see all the transactions done against an account number.
   3. A currency conversion system using the details in 'currencyDetails' lookup table. Upon performing a deposit/withdrawal, the user is provided an option to select from a list of predefined currency types. Upon the currency selection, the UI will be displaying the conversion rate to US Dollar to avoid confusion. This feature is available for single deposit/withdrawal and, not for bulk updating.
2. The bulk update feature does prove that the concurrency is handled as each of the transactions is looped and API call is made for each of the transaction.
3. Proper If... Else conditions are used wherever necessary.
4. Proper try... catch is implemented for all the DB related operations, in addition to the ChangeConflictException on ‘accountBalance’ update.

### Biggest Challenges

1. As I have no previous experience with WebApi, there was a learning curve to the technology. But, the knowledge and experience in Http Request in ASP.NET (using Http Handlers) and NODE.js helped minimise the learning phase.
2. A solution for concurrency was not required for any of the projects I have previously worked for. The first solution that came up was the traditional pessimistic way of using SQL LOCK. So, further research had to be done to done to determine the possibility of implementing an SQL LOCK on particular table row/cell instead of an entire table. This still felt like a pessimistic solution, as some of the transaction would still have to remain in a queue until the row/cell lock is released. Further research helped me understand about ChangeConflictException which sounded more optimistic. Which enabled the option for retrying a transaction until the account balance would go below 0.
3. The biggest challenge on top of the above was the time I could spend on this POC despite the busy schedule at my workplace. I am currently handling 4 projects. Out of the 4 projects, 3 of them are involving different team members where I have to both manage the team as well as deal with different customer. And, in the remaining one project, I am the only team member and point of contact for the customer. During the implementation time, I also had to travel out of state for 3 days to have meetings with customers. In short, what I have done is a result of a couple of sleepless nights. Please do consider this, if the proposed solution and/or the implementation/coding standards, is not up to your expectations.

### WebApi Details

The following are the details of API

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Method** | Balance | **HTTP Type** | | GET |
| An endpoint used to retrieve account balance.  Example URL: <http://localhost/api/account/balance> | | | | |
| Parameter Name | Data Type | Required | Outputs | |
| AccountNumber | Integer | Yes | AccountBalance (decimal)  AccountNumber (int)  HolderName (string)  ModifiedOn (datetime) | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Method** | All Accounts | **HTTP Type** | | GET |
| An endpoint used to retrieve account balance.  Example URL: [http://localhost/api/account/getallaccounts](http://localhost/api/account/balance) | | | | |
| Parameter Name | Data Type | Required | Outputs | |
|  |  |  | ***List of***  AccountBalance (decimal)  AccountNumber (int)  HolderName (string)  ModifiedOn (datetime) | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Method** | Deposit | **HTTP Type** | | POST |
| An endpoint to deposit funds in an account.  Example URL: <http://localhost/api/account/deposit> | | | | |
| Parameter Name | Data Type | Required | Outputs | |
| AccountNumber | Integer | Yes | AccountNumber (int)  Successful (bool)  Balance (decimal)  Currency (string)  Message (string)  ModifiedOn (datetime) | |
| Amount | Decimal | Yes |
| CurrencyId | Integer | Yes |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Method** | Withdraw | **HTTP Type** | | POST |
| A n endpoint used to withdraw funds from an account.  Example URL: http://localhost/account/withdraw | | | | |
| Parameter Name | Data Type | Required | Outputs | |
| AccountNumber | Integer | Yes | AccountNumber (int)  Successful (bool)  Balance (decimal)  Currency (string)  Message (string)  ModifiedOn (datetime) | |
| Amount | Decimal | Yes |
| CurrencyId | Integer | Yes |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Method** | All Currencies | **HTTP Type** | | GET |
| A n endpoint used to withdraw funds from an account.  Example URL: http://localhost/currency/getallcurrencies | | | | |
| Parameter Name | Data Type | Required | Outputs | |
|  |  |  | ***List Of***  CurrencyId (int)  CurrencyCode (string)  CurrencyName (string)  ConversionRateToDollar (decimal) | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Service Method** | All Transactions | **HTTP Type** | | GET |
| A n endpoint used to withdraw funds from an account.  Example URL: http://localhost/transaction/getalltransactionbyaccountnumber | | | | |
| Parameter Name | Data Type | Required | Outputs | |
| AccountNumber | Integer | Yes | ***List Of***  TransactionId (int)  AccountNumber (int)  CurrencyCode (string)  ConversionRateToDollar (decimal)  BalanceBefore (decimal)  BalanceAfter (decimal)  TransactionType (string)  TransactionAmount (decimal)  AddedOn (datetime)  AddedBy (string) | |