Currency Exchange Rate Converter Using C#

1. Project Overview

This project outlines the development of a currency exchange rate converter application using C#. The converter will allow users to input an amount in one currency and convert it to another currency using current or historical exchange rates.

2. Objectives

- Create a functional currency converter with a user-friendly interface
- Implement accurate currency conversion calculations
- Support multiple world currencies
- Optionally connect to live exchange rate APIs
- Ensure proper input validation

3. Variables Involved

Input Variables:

1. **Amount** (decimal) - The monetary value to be converted
2. **Source Currency** (string/enum) - Currency to convert from (e.g., USD, EUR, GBP)
3. **Target Currency** (string/enum) - Currency to convert to
4. **Exchange Rate Date** (DateTime) - Optional field for historical conversions

Calculation Variables:

1. **Exchange Rate** (decimal) - Current conversion rate between currencies
2. **Conversion Result** (decimal) - Calculated converted amount

Configuration Variables:

1. **Currency Database** (Dictionary/List) - Stores currency codes and names
2. **Exchange Rate Source** (API/local storage) - Source for current rates

4. Technical Implementation

Proposed Class Structure:

```
public class CurrencyConverter
{
    // Properties
    public decimal Amount { get; set; }
    public string SourceCurrency { get; set; }
    public string TargetCurrency { get; set; }
    public DateTime? RateDate { get; set; }

    // Exchange rate provider (could be API or local database)
    private IExchangeRateProvider _rateProvider;

    public CurrencyConverter(IExchangeRateProvider rateProvider)
    {
        _rateProvider = rateProvider;
    }

    // Methods
    public decimal Convert()
    {
        decimal rate = _rateProvider.GetExchangeRate(SourceCurrency,
TargetCurrency, RateDate);
        return Amount * rate;
    }

    public static Dictionary<string, string> GetAvailableCurrencies()
    {
        return new Dictionary<string, string>
        {
            {"USD", "US Dollar"},
            {"EUR", "Euro"},
            {"GBP", "British Pound"},
            {"JPY", "Japanese Yen"},
            // Add more currencies as needed
        };
    }
}

public interface IExchangeRateProvider
{
    decimal GetExchangeRate(string fromCurrency, string toCurrency, DateTime?
```

```
date);
}
```

Exchange Rate Data Options:

1. **Fixed Rates**:
    a. Hardcoded rates for development/testing
    b. Simple to implement but not up-to-date
2. **Local JSON/XML File**:
    a. Rates stored in a local file
    b. Can be updated periodically
3. **Web API Connection**:
    a. Real-time rates from services like:
        i. European Central Bank
        ii. Open Exchange Rates
        iii. Currency Layer
        iv. Fixer.io

5. User Interface Options

Console Application Version:

```
static void Main(string[] args)
{
    Console.WriteLine("Currency Converter");
    Console.WriteLine("Available currencies:");

    foreach (var currency in CurrencyConverter.GetAvailableCurrencies())
    {
        Console.WriteLine($"{currency.Key} - {currency.Value}");
    }

    Console.Write("Enter amount: ");
    decimal amount = decimal.Parse(Console.ReadLine());

    Console.Write("From currency (code): ");
    string from = Console.ReadLine().ToUpper();

    Console.Write("To currency (code): ");
    string to = Console.ReadLine().ToUpper();
```

```
    // Using fixed rates for demonstration
    var converter = new CurrencyConverter(new FixedRateProvider());
    decimal result = converter.Convert();

    Console.WriteLine($"{amount} {from} = {result:F2} {to}");
}
```

Windows Forms/WPF Features:

- Dropdown lists for currency selection
- Numeric input for amount
- Date picker for historical rates
- Swap currencies button
- Favorites/recents section

6. Validation Requirements

- Validate currency codes exist
- Ensure amount is positive number
- Handle API connection errors gracefully
- Manage currency precision (different currencies have different decimal places)

7. Development Timeline

1. Core logic implementation - 2 days
2. Basic UI development - 2 days
3. API integration (if used) - 2 days
4. Testing and validation - 1 day
5. Documentation - 1 day

8. Future Enhancements

- Add currency charts/graphs
- Implement offline mode with cached rates
- Add currency conversion history

- Support cryptocurrency conversions
- Create watchlists for favourite currency pairs
- Implement currency rate alert