# Small Stocktaking APP


MAY 4

**COMPANY NAME**

**Authored by: Your Name**

Logo Name

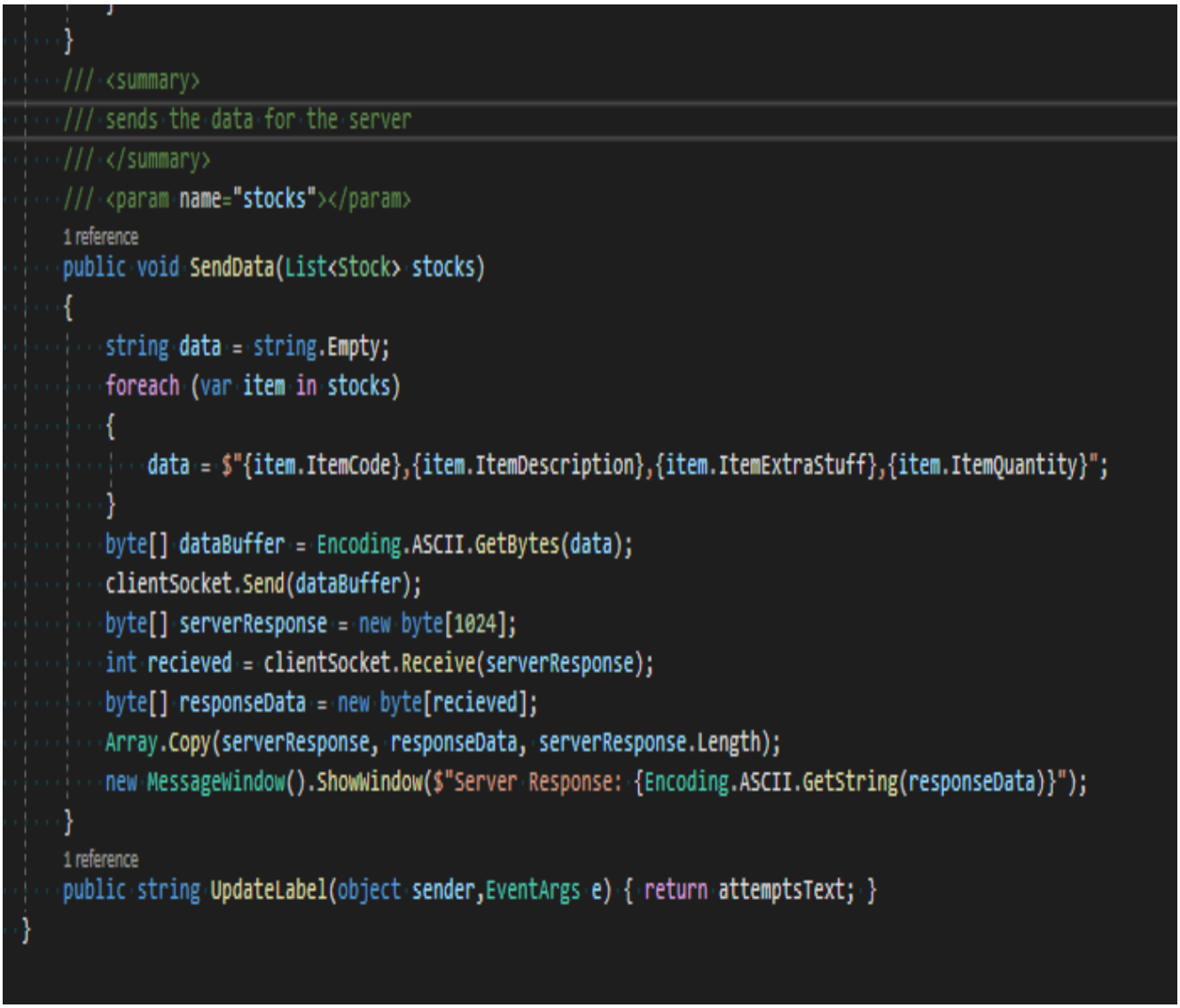
# Contents

# Client Controller

```
using System;
using System.Net;
using TextHandler;
using ErrorHandler;
using System.Net.Sockets;
using System.Collections.Generic;
using System.Text;
using CustomControls;
namespace NetworkLayer
{
    2 references
    public class ClientController
    {
        private Socket clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        private int port = 100;
        private int ConnectAttempts = 5;
        private string attemptsText;
        /// <summary>
        /// this method handels the connection between a server and a client
        /// </summary>
        /// <param name="stocks">the blueprint for stocks</param>
        1 reference
        public void ConnectTOSever(List<Stock> stocks)
        {
            int attempts = 0;
            while (!clientSocket.Connected)
            {
                try
                {
                    if (attempts < ConnectAttempts)
                    {
                        attempts++;
                        clientSocket.Connect(new IPEndPoint(IPAddress.Loopback, port));
                        new StatusUpdater().eventf += UpdateLabel;
                    }
                    else
                    {
                        break;
```

```
2 references
public class ClientController
{
    private Socket clientSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
    private int port = 100;
    private int ConnectAttempts = 5;
    private string attemptsText;
    /// <summary>
    /// this method handels the connection between a server and a client
    /// </summary>
    /// <param name="stocks">the blueprint for stocks</param>
    1 reference
    public void ConnectTOSever(List<Stock> stocks)
    {
        int attempts = 0;
        while (!clientSocket.Connected)
        {
            try
            {
                if (attempts < ConnectAttempts)
                {
                    attempts++;
                    clientSocket.Connect(new IPEndPoint(IPAddress.Loopback, port));
                    new StatusUpdater().eventf += UpdateLabel;
                }
                else
                {
                    break;
                }
            }
            catch (SocketException ex)
            {
                attemptsText = $"Connected Sockets: {attempts}";
                ErrorHandlerClass.WriteLog(ex,Environment.UserName,DateTime.Now);
            }
        }
        if (clientSocket.Connected)
```

```
1 reference
public void ConnectTOSever(List<Stock> stocks)
{
    int attempts = 0;
    while (!clientSocket.Connected)
    {
        try
        {
            if (attempts < ConnectAttempts)
            {
                attempts++;
                clientSocket.Connect(new IPEndPoint(IPAddress.Loopback, port));
                new StatusUpdater().eventf += UpdateLabel;
            }
            else
            {
                break;
            }
        }
        catch (SocketException ex)
        {
            attemptsText = $"Connected Sockets: {attempts}";
            ErrorHandlerClass.WriteLog(ex,Environment.UserName,DateTime.Now);
        }
    }
    if (clientSocket.Connected)
    {
        SendData(stocks);
        new MessageWindow().ShowWindow("Connected to server, Sending Data");
    }
    else if (!clientSocket.Connected)
    {
        new MessageWindow().ShowWindow("Connection Failed");
    }
}
```



```
    }
    /// <summary>
    /// sends the data for the server
    /// </summary>
    /// <param name="stocks"></param>
    1 reference
    public void SendData(List<Stock> stocks)
    {
        string data = string.Empty;
        foreach (var item in stocks)
        {
            data = $"{item.ItemCode},{item.ItemDescription},{item.ItemExtraStuff},{item.ItemQuantity}";
        }
        byte[] dataBuffer = Encoding.ASCII.GetBytes(data);
        clientSocket.Send(dataBuffer);
        byte[] serverResponse = new byte[1024];
        int recieved = clientSocket.Receive(serverResponse);
        byte[] responseData = new byte[recieved];
        Array.Copy(serverResponse, responseData, serverResponse.Length);
        new MessageWindow().ShowWindow($"Server Response: {Encoding.ASCII.GetString(responseData)}");
    }
    1 reference
    public string UpdateLabel(object sender,EventArgs e) { return attemptsText; }
}
```

# Client view

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using TextHandler;
using NetworkLayer;
using CustomControls;
using System.Drawing;
namespace Oom_Chris_App
{
    5 references
    public partial class ClientForm : Form
    {
        public static ClientForm instance;
        private List<Stock> stocks;
        1 reference
        public ClientForm()
        {
            InitializeComponent();
            instance = this;
        }
        1 reference
        private void btnSubmit_Click(object sender, EventArgs e)
        {
            textWriterAndReader writer = new textWriterAndReader();
            string path = saveFile();
            if(path != string.Empty)
                writer.WriteData(path, stocks);
            new MessageWindow().ShowWindow("Connecting to Server.....");
            ClientController clientController = new ClientController();
            clientController.ConnectTOSever(stocks);
            lbConnection.Text = new StatusUpdater().UpdateLabel();
        }
        1 reference
        private void btnImport_Click(object sender, EventArgs e)
        {
            textWriterAndReader Reader = new textWriterAndReader();
            OpenFileDialog fileDialog = new OpenFileDialog();
            if (fileDialog.ShowDialog() == DialogResult.OK)
                stocks = Reader.ReadData(fileDialog.FileName);
```

```
1 reference
private void btnImport_Click(object sender, EventArgs e)
{
    textWriterAndReader Reader = new textWriterAndReader();
    OpenFileDialog fileDialog = new OpenFileDialog();
    if (fileDialog.ShowDialog() == DialogResult.OK)
        stocks = Reader.ReadData(fileDialog.FileName);
    dgvMainView.DataSource = stocks;
    initializeDataGrid();
}
1 reference
private void dgvMainView_CellValueChanged(object sender, DataGridViewCellEventArgs e)
{
    UpdateDataSource(dgvMainView.CurrentRow.Cells["ItemQuantity"].EditedFormattedValue.ToString());
    dgvMainView.CommitEdit(DataGridViewDataErrorContexts.Commit);
}
/// <summary>
/// this method save a textfile copy of the data on the local machine
/// </summary>
/// <returns>path</returns>
1 reference
private string saveFile()//maybe meer robust maak TEST
{
    SaveFileDialog fileDialog = new SaveFileDialog();
    fileDialog.Title = "SaveTextFile";
    fileDialog.FileName = "NewDataFile";
    fileDialog.DefaultExt = ".txt";
    fileDialog.AddExtension = true;
    fileDialog.Filter = "Text Files (*.txt)|";
    return fileDialog.ShowDialog() == DialogResult.OK ? fileDialog.FileName : string.Empty;
}
/// <summary>
/// this method commit changes to the datasource
/// </summary>
```

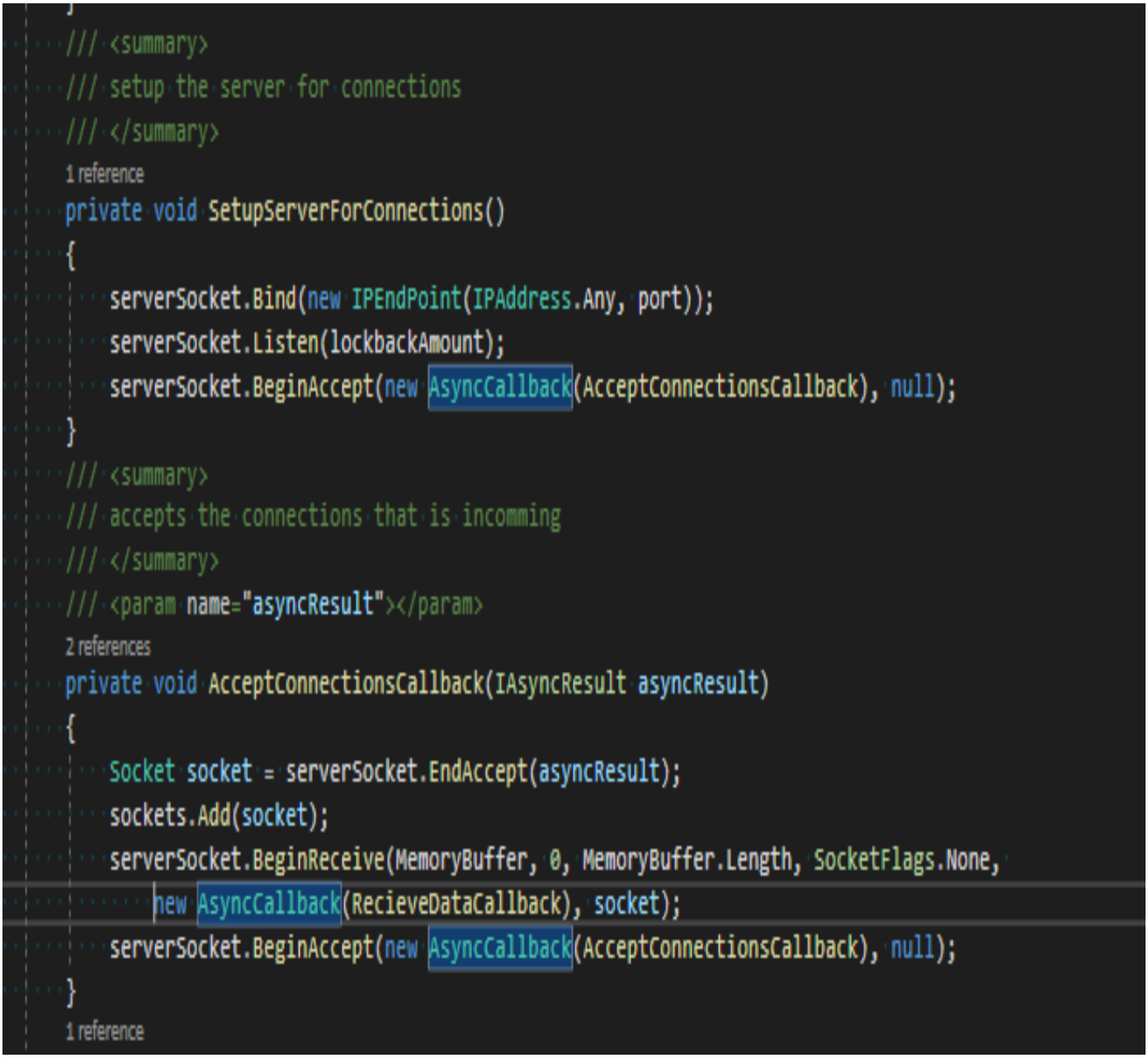
```
    }
    /// <summary>
    /// this method commit changes to the datasource
    /// </summary>
    /// <param name="value"></param>
    1 reference
    private void UpdateDataSource(string value)
    {
        value = value == string.Empty || value == null ? "null" : value;
        List<Stock> newStock = new List<Stock>();
        foreach (var item in stocks)
        {
            if (dgvMainView.CurrentRow.Cells["ItemCode"].Value.ToString() == item.ItemCode)
            {
                Stock replacement = new Stock()
                {
                    ItemCode = item.ItemCode,
                    ItemDescription = item.ItemDescription,
                    ItemExtraStuff = item.ItemExtraStuff,
                    ItemQuantity = value
                };
                if (stocks.Contains(item))
                {
                    int index = stocks.IndexOf(item);
                    stocks.Remove(item);
                    stocks.Insert(index,replacement);
                }
                break;
            }
        }
        dgvMainView.DataSource = stocks;
        initializeDataGrid();
    }
    /// <summary>
    /// this method initializes the datagrid to make most columns grey and readonly except for quantity
    /// </summary>

    private void initializeDataGrid()
```

```
                break;
            }
        }
        dgvMainView.DataSource = stocks;
        initializeDataGrid();
    }
    /// <summary>
    /// this method initializes the datagrid to make most columns grey and readonly except for quantity
    /// </summary>
    2 references
    private void initializeDataGrid()
    {
        dgvMainView.Columns["itemCode"].ReadOnly = true;
        dgvMainView.Columns["itemExtraStuff"].ReadOnly = true;
        dgvMainView.Columns["itemDescription"].ReadOnly = true;
        dgvMainView.Columns["itemCode"].DefaultCellStyle.BackColor = Color.Gray;
        dgvMainView.Columns["itemExtraStuff"].DefaultCellStyle.BackColor = Color.Gray;
        dgvMainView.Columns["itemDescription"].DefaultCellStyle.BackColor = Color.Gray;
    }
}
}
```

# Server controller

```
using System;
using System.Net;
using TextHandler;
using System.Text;
using CustomControls;
using System.Net.Sockets;
using System.Collections.Generic;
namespace NetworkLayer
{
    1 reference
    public class ServerController
    {
        private Socket serverSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        private int port = 100;
        private int lockbackAmount = 5;
        private byte[] MemoryBuffer = new byte[100];
        private List<Stock> stocks = new List<Stock>();
        private List<object> listOfList = new List<object>();
        private static List<Socket> sockets = new List<Socket>();
        /// <summary>
        /// handles all clients that needs to connect to server
        /// </summary>
        /// <returns></returns>
        1 reference
        public List<object> GetConnections()
        {
            SetupServerForConnections();
            new MessageWindow().ShowWindow($"Connected Clients: {sockets.Count}");
            return listOfList;
        }
        1 reference
```



```
    /// <summary>
    /// setup the server for connections
    /// </summary>
    1 reference
    private void SetupServerForConnections()
    {
        serverSocket.Bind(new IPEndPoint(IPAddress.Any, port));
        serverSocket.Listen(lockbackAmount);
        serverSocket.BeginAccept(new AsyncCallback(AcceptConnectionsCallback), null);
    }
    /// <summary>
    /// accepts the connections that is incomming
    /// </summary>
    /// <param name="asyncResult"></param>
    2 references
    private void AcceptConnectionsCallback(IAsyncResult asyncResult)
    {
        Socket socket = serverSocket.EndAccept(asyncResult);
        sockets.Add(socket);
        serverSocket.BeginReceive(MemoryBuffer, 0, MemoryBuffer.Length, SocketFlags.None,
            new AsyncCallback(RecieveDataCallback), socket);
        serverSocket.BeginAccept(new AsyncCallback(AcceptConnectionsCallback), null);
    }
    1 reference
```

```
    /// <summary>
    /// recieve request from clients
    /// </summary>
    /// <param name="asyncResult"></param>
    1 reference
    private void RecieveDataCallback(IAsyncResult asyncResult)
    {
        Socket socket = (Socket)asyncResult.AsyncState;
        int recieved = socket.EndReceive(asyncResult);
        byte[] bufferArray = new byte[recieved];
        Array.Copy(MemoryBuffer, bufferArray, recieved);
        string recievedText = Encoding.ASCII.GetString(bufferArray);
        string[] data = recievedText.Split(',');
        stocks = new List<Stock>();//maybe check die deel
        for (int i =0; i < data.Length; i++)
        {
            Stock stock = new Stock() { ItemCode = data[i], ItemDescription = data[i+1],
                ItemExtraStuff = data[i+2], ItemQuantity = data[i+3] };
            i += 3;
            stocks.Add(stock);
        }
        listOfList.Add(stocks);
        string recievedResponse = string.Empty;
        recievedResponse = recievedText == string.Empty || recievedText == null ? "Invalid Response" : "data Recieved";
        byte[] dataToSend = Encoding.ASCII.GetBytes(recievedResponse);
        socket.BeginSend(dataToSend, 0, dataToSend.Length, SocketFlags.None, new AsyncCallback(SendDataCallBack), null);
    }
    /// <summary>
    /// send data back
    /// </summary>
    /// <param name="asyncResult"></param>
    1 reference
    private void SendDataCallBack(IAsyncResult asyncResult)
    {
```

```
    }
    /// <summary>
    /// send data back
    /// </summary>
    /// <param name="asyncResult"></param>
    1 reference
    private void SendDataCallBack(IAsyncResult asyncResult)
    {
        Socket socket = (Socket)asyncResult.AsyncState;
        socket.EndSend(asyncResult);
    }
}
```

# Server view

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using TextHandler;
using NetworkLayer;
using CustomControls;
using System.Drawing;
namespace Oom_Chris_App
{
    4 references
    public partial class ServerView : Form
    {
        private List<Stock> stocksList;
        private List<Stock> stocksHolder;
        private List<object> listOfList;
        1 reference
        public ServerView()
        {
            InitializeComponent();
        }
        1 reference
        private void btnGetData_Click(object sender, EventArgs e)
        {
            listOfList = new ServerController().GetConnections();
            if(listOfList!=null)
                ProcessStocks();
            if (stocksList != null)
                dgvServer.DataSource = stocksList;
            if (dgvServer.DataSource != null)
                initializeDataGrid();
        }
        1 reference
        private void btnNull_Click(object sender, EventArgs e)//test
        {
            for (int i = 0; i < dgvServer.Rows.Count; i++)
            {
                string value = dgvServer.Rows[i].Cells["ItemQuantity"].Value.ToString();
                if (value == null || value == string.Empty)
                {
```

```
1 reference
private void btnNull_Click(object sender, EventArgs e)//test
{
    for (int i = 0; i < dgvServer.Rows.Count; i++)
    {
        string value = dgvServer.Rows[i].Cells["ItemQuantity"].Value.ToString();
        if (value == null || value == string.Empty)
        {
            foreach(var item in stocksList)
            {
                if (dgvServer.Rows[i].Cells["ItemCode"].Value.ToString() == item.ItemCode)
                {
                    Stock replacement = new Stock()
                    {
                        ItemCode = item.ItemCode,
                        ItemDescription = item.ItemDescription,
                        ItemExtraStuff = item.ItemExtraStuff,
                        ItemQuantity = "0"
                    };
                    if (stocksList.Contains(item))
                    {
                        int index = stocksList.IndexOf(item);
                        stocksList.Remove(item);
                        stocksList.Insert(index, replacement);
                    }
                    break;
                }
            }
        }
    }
    if (stocksList != null)
        dgvServer.DataSource = stocksList;
    if(dgvServer.DataSource!=null)
        initializeDataGrid();
    new MessageWindow().ShowWindow("Remember to Export to TextFile to save this new Data");
```

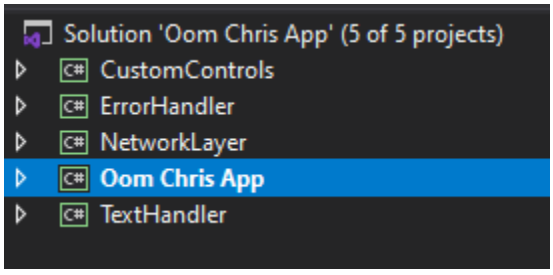
```
        }
        if (stocksList != null)
            dgvServer.DataSource = stocksList;
        if(dgvServer.DataSource!=null)
            initializeDataGrid();
        new MessageWindow().ShowWindow("Remember to Export to TextFile to save this new Data");
    }
    1 reference
    private void btnExport_Click(object sender, EventArgs e)
    {
        textWriterAndReader writer = new textWriterAndReader();
        string path = saveFile();
        if (path != string.Empty)
            writer.WriteData(path, stocksList);
    }
    /// <summary>
    /// this method save a textfile copy of the data on the local machine
    /// </summary>
    /// <returns>path</returns>
    1 reference
    private string saveFile()//maybe meer robust maak TEST
    {
        SaveFileDialog fileDialog = new SaveFileDialog();
        fileDialog.Title = "SaveTextFile";
        fileDialog.FileName = "NewDataFile";
        fileDialog.DefaultExt = ".txt";
        fileDialog.AddExtension = true;
        fileDialog.Filter = "Text Files (*.txt)|";
        return fileDialog.ShowDialog() == DialogResult.OK ? fileDialog.FileName : string.Empty;
    }
    /// <summary>
    /// this method initializes the datagrid to make most columns grey and readonly except for quantity
    /// </summary>
    2 references
    private void initializeDataGrid()
    {
        dgvServer.Columns["itemCode"].ReadOnly = true;
```

```
2 references
private void initializeDataGrid()
{
    dgvServer.Columns["itemCode"].ReadOnly = true;
    dgvServer.Columns["itemExtraStuff"].ReadOnly = true;
    dgvServer.Columns["itemDescription"].ReadOnly = true;
    dgvServer.Columns["itemCode"].DefaultCellStyle.BackColor = Color.Gray;
    dgvServer.Columns["itemExtraStuff"].DefaultCellStyle.BackColor = Color.Gray;
    dgvServer.Columns["itemDescription"].DefaultCellStyle.BackColor = Color.Gray;
}
/// <summary>
/// this method processes stock that are the same and calculates the stock that correspond quantity values together
/// </summary>
1 reference
private void ProcessStocks()//maak progressBar vir die
{
    if(listOfList != null)
    {
        foreach (var item in listOfList)
        {
            stocksHolder = new List<Stock>();
            foreach (var element in (List<Stock>)item)
            {
                stocksHolder.Add(element);
                foreach (var stocks in stocksHolder)
                {
                    if (stocks.ItemCode == element.ItemCode)
                    {
                        Stock stock1 = stocks;
                        stock1.ItemQuantity = string.Empty;
                        stock1.ItemQuantity = (int.Parse(stocks.ItemQuantity) + int.Parse(element.ItemQuantity)).ToString();
                        stocksList.Add(stock1);
                    }
                }
            }
        }
        foreach (var item in listOfList)
```

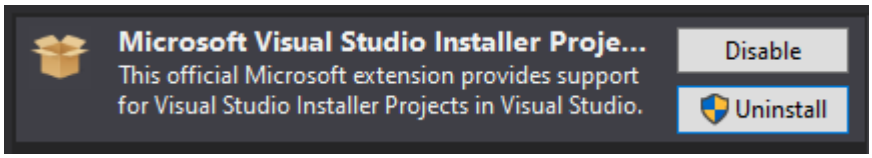
```
        private void ProcessStocks()//maak progressBar vir die
        {
            if(listOfList != null)
            {
                foreach (var item in listOfList)
                {
                    stocksHolder = new List<Stock>();
                    foreach (var element in (List<Stock>)item)
                    {
                        stocksHolder.Add(element);
                        foreach (var stocks in stocksHolder)
                        {
                            if (stocks.ItemCode == element.ItemCode)
                            {
                                Stock stock1 = stocks;
                                stock1.ItemQuantity = string.Empty;
                                stock1.ItemQuantity = (int.Parse(stocks.ItemQuantity) + int.Parse(element.ItemQuantity)).ToString();
                                stocksList.Add(stock1);
                            }
                        }
                    }
                }
                foreach (var item in listOfList)
                {
                    foreach (var element in (List<Stock>)item)
                    {
                        if (!stocksList.Contains(element))
                        {
                            stocksList.Add(element);
                        }
                    }
                }
            }
        }
    }
}
```

```
using System.Collections.Generic;
using System.IO;
namespace TextHandler
{
    6 references
    public class textWriterAndReader
    {
        private List<Stock> stockList = new List<Stock>();
        /// <summary>
        /// this method reads all data from a textfile that is imported into app
        /// </summary>
        /// <param name="path"></param>
        /// <returns></returns>
        1 reference
        public List<Stock> ReadData(string path)
        {
            Stock stock;
            using(StreamReader reader = new StreamReader(path))
            {
                while (!reader.EndOfStream)
                {
                    stock = new Stock
                    {
                        ItemCode = reader.ReadLine(),
                        ItemDescription = reader.ReadLine(),
                        ItemExtraStuff = reader.ReadLine(),
                    };
                    stockList.Add(stock);
                }
            }
            return stockList;
        }
        /// <summary>
```

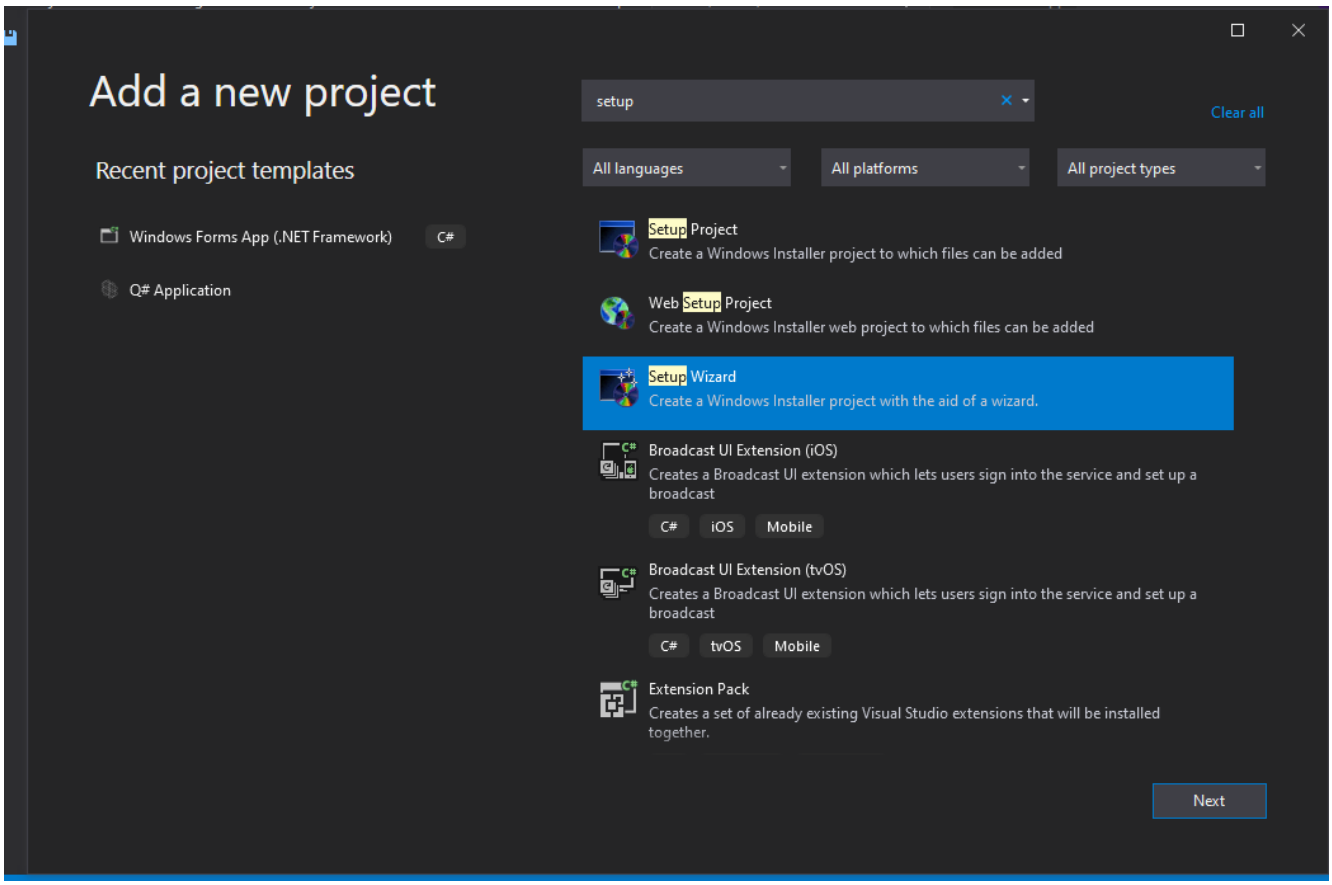
```
    }
    /// <summary>
    /// this method writes all data to a text file of the stocks class
    /// </summary>
    /// <param name="path"></param>
    /// <param name="stocks"></param>
    2 references
    public void WriteData(string path,List<Stock> stocks)
    {
        using(StreamWriter writer = new StreamWriter(path))
        {
            foreach(var item in stocks)
            {
                writer.WriteLine(item.ItemCode);
                writer.WriteLine(item.ItemDescription);
                writer.WriteLine(item.ItemExtraStuff);
                writer.WriteLine(item.ItemQuantity);
            }
        }
    }
}
```

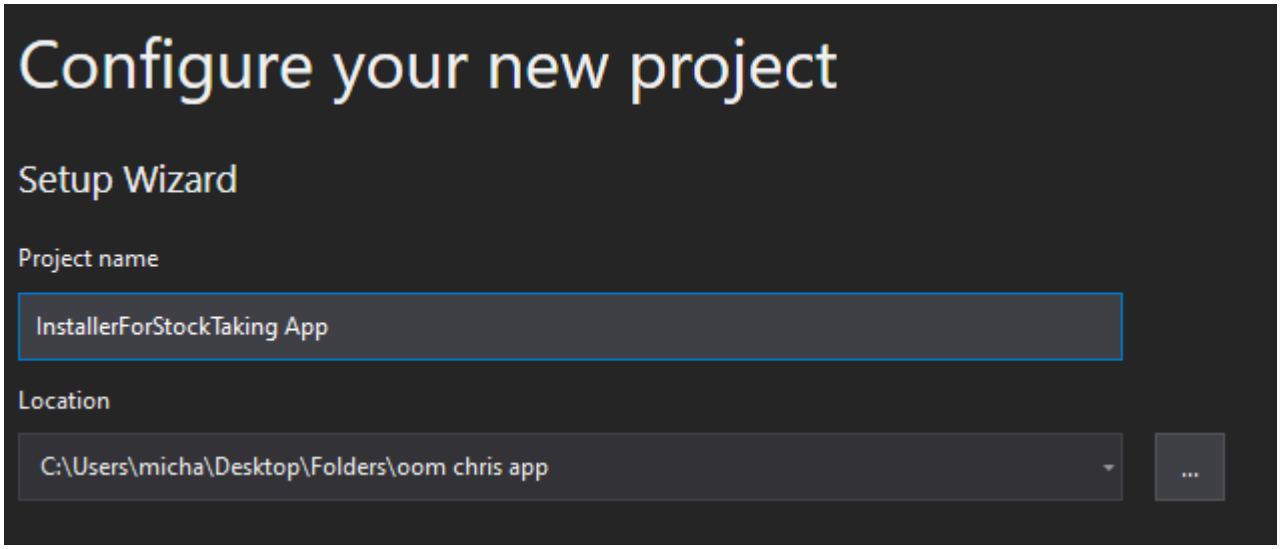


# Creating A installer for the app




***Above the extension for creating old school installer***






**Do you want to create a setup program to install an application?**

- (•) Create a setup for a Windows application
- ( ) Create a setup for a web application

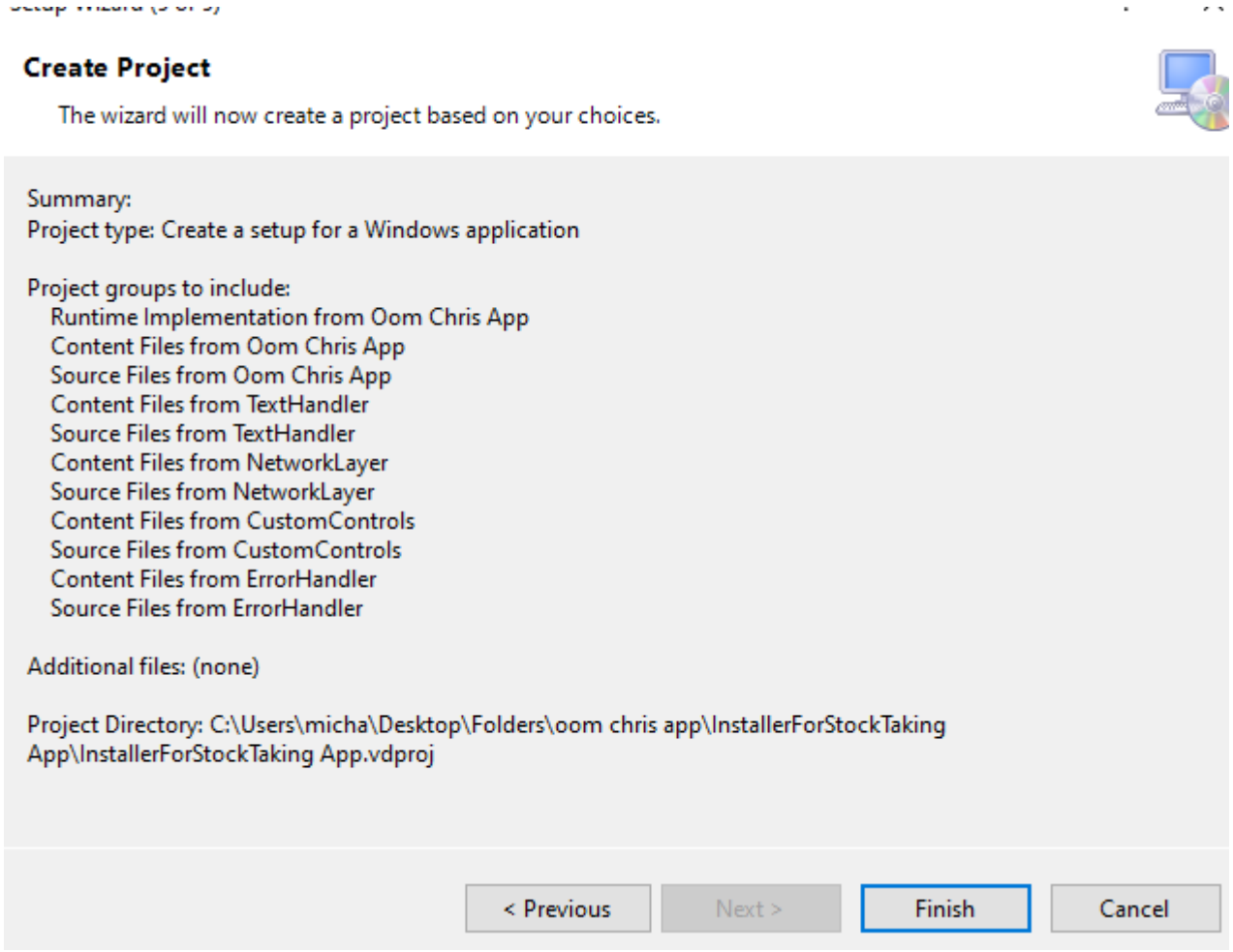
**Do you want to create a redistributable package?**

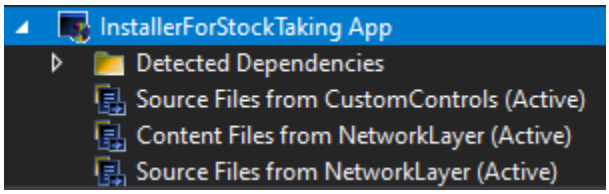
- ( ) Create a merge module for Windows Installer
- ( ) Create a downloadable CAB file

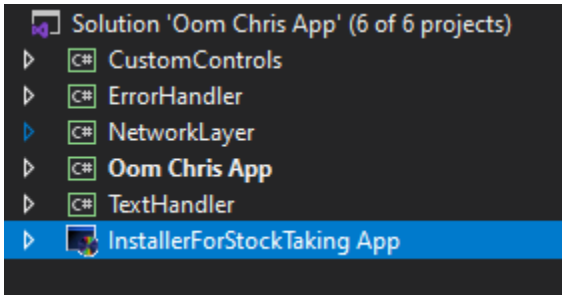
**Which project output groups do you want to include?**

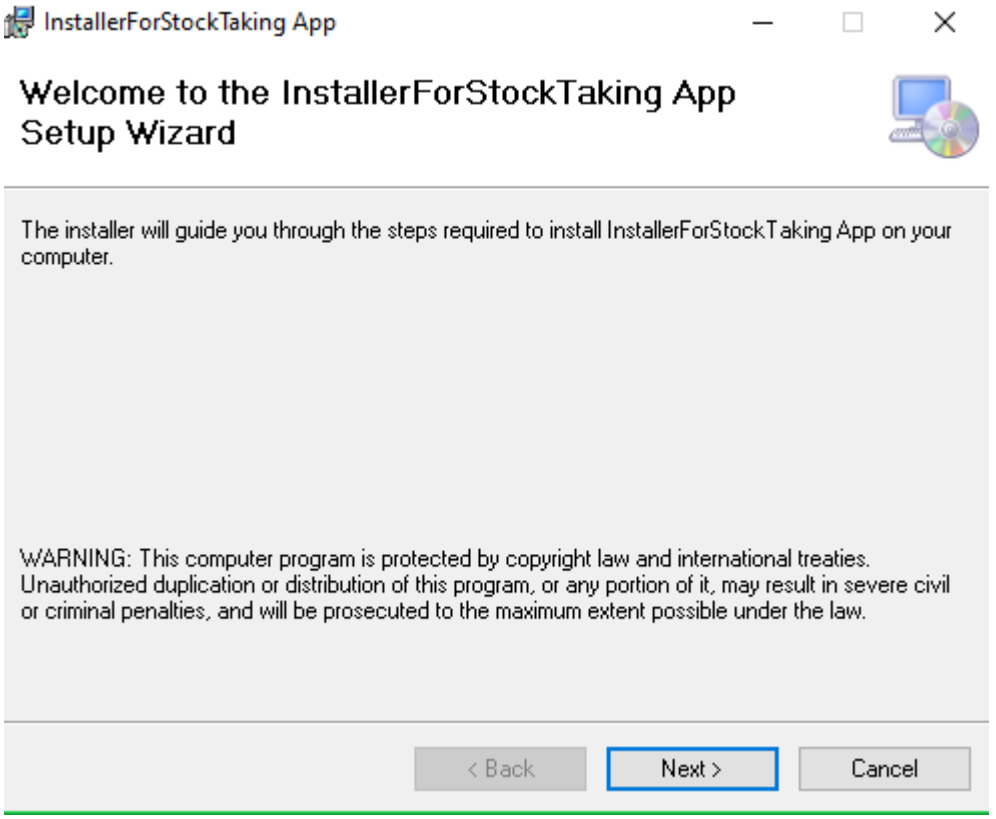
- [ ] Locally-Copied Items from Oom Chris App
- [x] Runtime Implementation from Oom Chris App
- [ ] Localized resources from Oom Chris App
- [x] Content Files from Oom Chris App
- [ ] XML Serialization Assemblies from Oom Chris App
- [ ] Primary output from Oom Chris App
- [x] Source Files from Oom Chris App
- [ ] Debug Symbols from Oom Chris App
- [ ] Documentation Files from Oom Chris App
- [ ] Locally-Copied Items from TextHandler
- [ ] Runtime Implementation from TextHandler

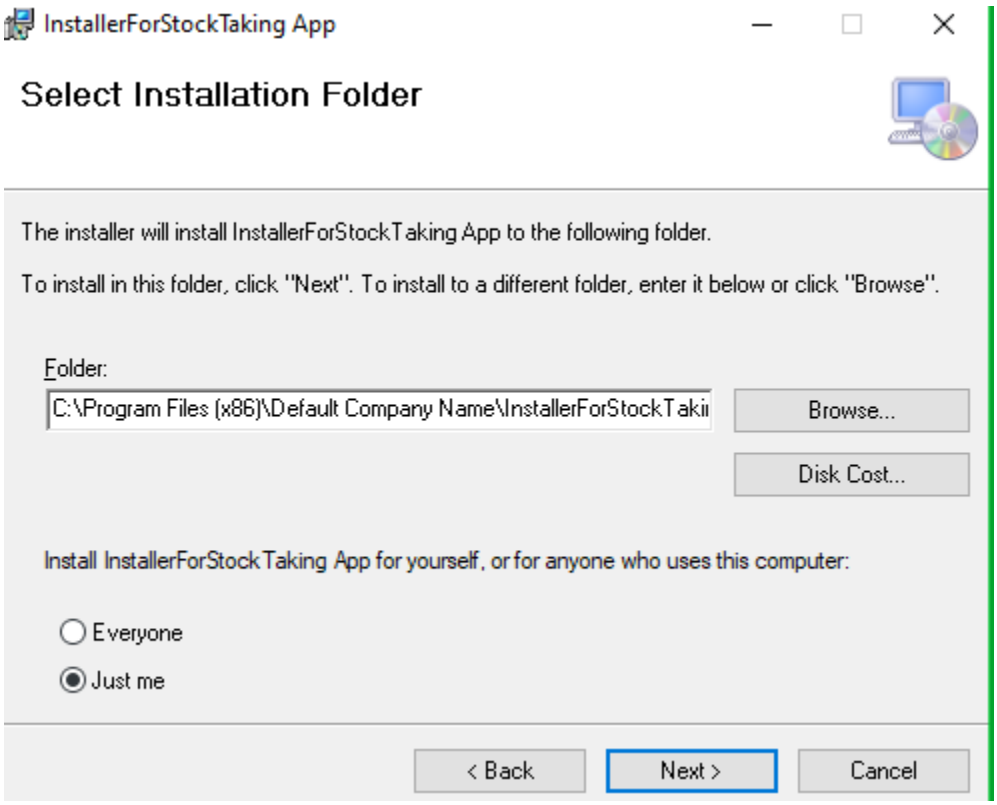
Description:
Contains the runtime implementation assemblies for the targeted framework.

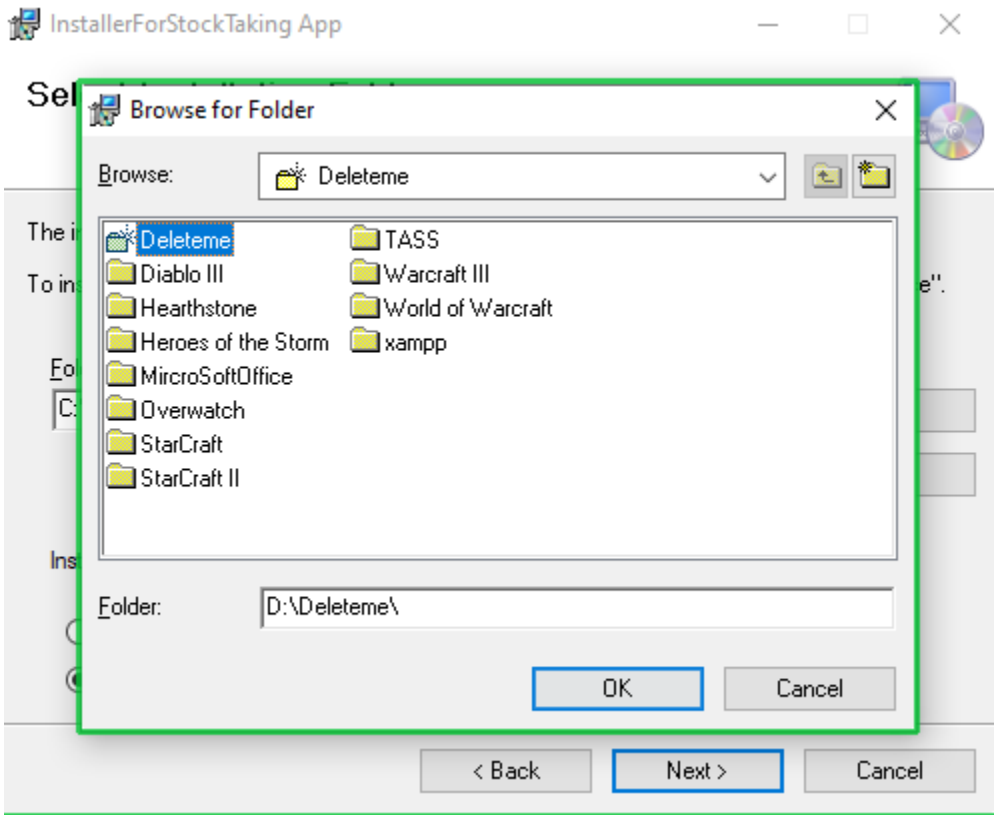

Create Project
The wizard will now create a project based on your choices.
Summary:
Project type: Create a setup for a Windows application
Project groups to include:
Runtime Implementation from Oom Chris App
Content Files from Oom Chris App
Source Files from Oom Chris App
Content Files from TextHandler
Source Files from TextHandler
Content Files from NetworkLayer
Source Files from NetworkLayer
Content Files from CustomControls
Source Files from CustomControls
Content Files from ErrorHandler
Source Files from ErrorHandler
Additional files: (none)
Project Directory: C:\Users\micha\Desktop\Folders\oom chris app\InstallerForStockTaking App\InstallerForStockTaking App.vdproj
< Previous
Next >
Finish
Cancel

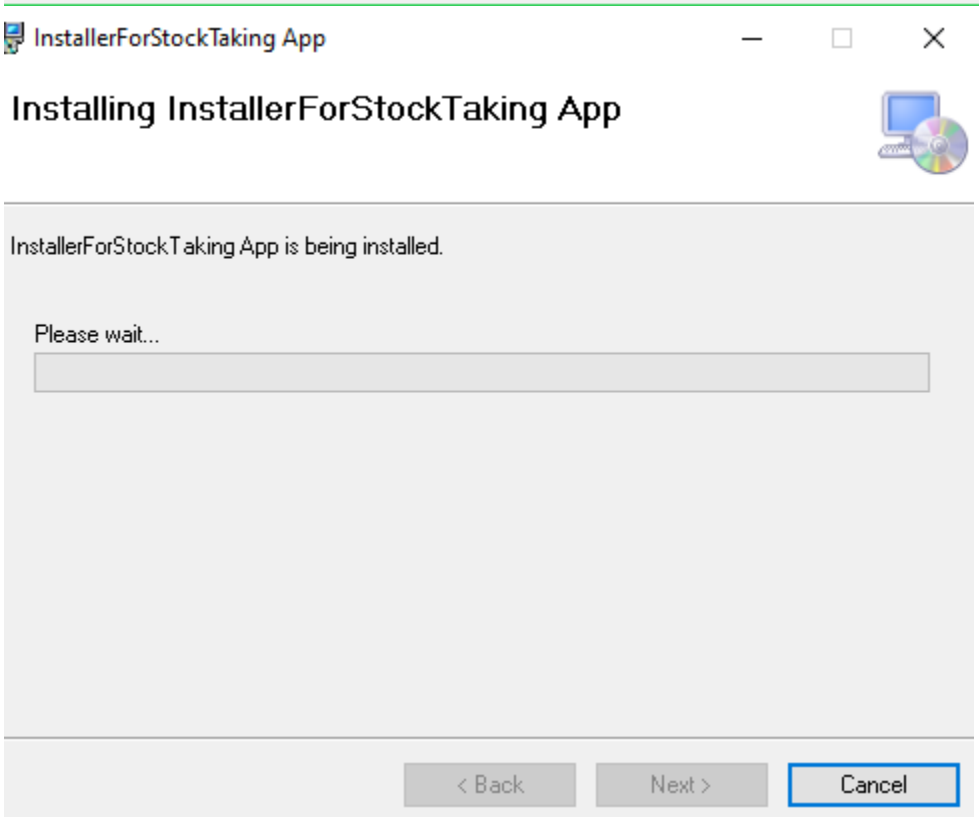

InstallerForStockTaking App
Detected Dependencies
Source Files from CustomControls (Active)
Content Files from NetworkLayer (Active)
Source Files from NetworkLayer (Active)


Solution 'Oom Chris App' (6 of 6 projects)
CustomControls
ErrorHandler
NetworkLayer
Oom Chris App
TextHandler
InstallerForStockTaking App


InstallerForStockTaking App
Welcome to the InstallerForStockTaking App Setup Wizard
The installer will guide you through the steps required to install InstallerForStockTaking App on your computer.
WARNING: This computer program is protected by copyright law and international treaties. Unauthorized duplication or distribution of this program, or any portion of it, may result in severe civil or criminal penalties, and will be prosecuted to the maximum extent possible under the law.
< Back
Next >
Cancel


InstallerForStockTaking App
Select Installation Folder
The installer will install InstallerForStockTaking App to the following folder.
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".
Folder:
C:\Program Files (x86)\Default Company Name\InstallerForStockTaki
Browse...
Disk Cost...
Install InstallerForStockTaking App for yourself, or for anyone who uses this computer:
Everyone
Just me
< Back
Next >
Cancel


InstallerForStockTaking App
Browse for Folder
Browse:
Deleteme
Deleteme
Diablo III
Hearthstone
Heroes of the Storm
MircroSoftOffice
Overwatch
StarCraft
StarCraft II
TASS
Warcraft III
World of Warcraft
xampp
Folder:
D:\Deleteme\
OK
Cancel
< Back
Next >
Cancel


InstallerForStockTaking App
Installing InstallerForStockTaking App
InstallerForStockTaking App is being installed.
Please wait...
< Back
Next >
Cancel