9/15/2016

Daniel Field (3107593512)

Charity Kitchen Technical Documentation

ICTPRG501 and ICTWEB503

Contents

[Pages 2](#_Toc461695836)

[Default.aspx 2](#_Toc461695837)

[Login.aspx 2](#_Toc461695838)

[Orders.aspx 3](#_Toc461695839)

[OrderEdit.aspx 3](#_Toc461695840)

[Meals.aspx 3](#_Toc461695841)

[Ingredients.aspx 3](#_Toc461695842)

[Users.aspx 3](#_Toc461695843)

[Roles.aspx 3](#_Toc461695844)

[Web Service 4](#_Toc461695845)

[Algorithms 4](#_Toc461695846)

[Error Handling 7](#_Toc461695847)

[Testing 8](#_Toc461695848)

[Grey Box Test 8](#_Toc461695849)

[White Box Test 10](#_Toc461695850)

[The Database 12](#_Toc461695851)

[Relationships 12](#_Toc461695852)

[Attributes 12](#_Toc461695853)

[Entity-related Classes 13](#_Toc461695854)

[Advantages and Limitations of HTTP-based Applications 14](#_Toc461695855)

[Design Patterns in the Microsoft .NET Framework 15](#_Toc461695856)

[Debugging Techniques 16](#_Toc461695857)

[Breakpoints 16](#_Toc461695858)

[Watches 17](#_Toc461695859)

[Error Messages 17](#_Toc461695860)

[Three Logging Frameworks Available for .NET 18](#_Toc461695861)

[SmartInspect 18](#_Toc461695862)

[NLog 18](#_Toc461695863)

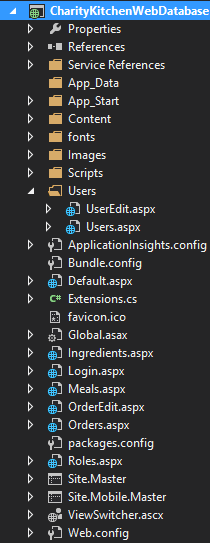
[Log4net 19](#_Toc461695864)

[Site Map 19](#_Toc461695865)

[References 20](#_Toc461695866)

# Pages

The main project within the system contains all of the files that are used to output information to the user’s screen. There are a variety of ASPX pages in the project.



All of these pages contain help sections that explain how each page works. Each page except for Login.aspx uses

## Default.aspx

This page is the first page that is seen when the web application starts. This page contains buttons that link to the other pages throughout the application.

The buttons will be only enabled if you have access to those pages. This is determined by the roles functionality.

## Login.aspx

This page is for logging in, it is where every other page redirects to when they do not receive valid credentials.

## Orders.aspx

This page is for adding/editing/deleting orders. This page is the most critical section of the application.

## OrderEdit.aspx

This page allows you to edit new orders, and orders that were previously created. It contains a list of order lines (recipes that the order consists of), and the ability to add/edit/delete order lines.

## Meals.aspx

The meals (recipes) page is used for listing, editing, and deleting recipes. You can select the recipes individually to edit the ingredients of which they consist.

## Ingredients.aspx

This page lists all of the ingredients in the database. These ingredients are referenced in the meals/recipes. Use this page to add/edit/delete ingredients which will be used in the recipes.

## Users.aspx

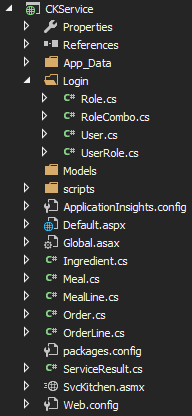
This webpage contains a list of all of the user accounts. You can add/edit/delete users here. The users have default role permissions set to zero, which denies access to all of the pages except for the home page.

## Roles.aspx

This page is used for adding, editing, and deleting roles. The data on this page controls which pages people can access.

# Web Service

The web service contains all of the code that interacts with the database.



## Algorithms

#### Get Orders(out ServiceResult result)

1. Create a list of orders
2. Connect to database
3. Select everything from tblOrders
4. Add each order to the list
5. Close connection
6. Create a new service result
7. Return the list as an array

#### GetOrderByID(int id, out ServiceResult result)

1. Create a new Order object
2. Connect to database
3. Get the first row in tblOrders where the primary key matches the specified integer
4. Read the data and put it in the order object
5. Close connection
6. Create service result
7. Return the order

#### addOrder(Order record, out ServiceResult result)

1. If the record id is zero
   1. Set the query to an insert query with the data from the record
2. Else
   1. Set the query to an update query with the data from the record
3. Connect to database
4. Run query
5. Create a service result

#### DeleteOrder(int id, out ServiceResult result)

1. Call method “DeleteOrderLineByOrderID” with the specified order primary key (this prevents there being order lines that are not related to any order)
2. Connect to db
3. Run delete query to delete the row with the specified primary key
4. Close connection
5. Create new ServiceResult

#### addOrderLine(OrderLine record, out ServiceResult result)

1. If the record id is zero
   1. Set the query to an insert query with the data from the record
2. Else
   1. Set the query to an update query with the data from the record
3. Connect to database
4. Run query
5. Create a service result

#### Get OrderLines(int id, out ServiceResult result)

1. Create a list of order lines
2. Connect to database
3. Select everything from tblOrderLines where the primary key equals the specified integer
4. Add each order line to the list
5. Close connection
6. Create a new service result
7. Return the list as an array

#### GetByOrderLineId(int id, out ServiceResult result)

1. Create a new OrderLine object
2. Connect to database
3. Get the first row in qryOrderLines where the primary key matches the specified integer
4. Read the data and put it in the order line object
5. Close connection
6. Create service result
7. Return the order line

#### DeleteOrderLine(int id, out ServiceResult result)

1. Connect to db
2. Run delete query to delete the specified order line
3. Close connection
4. Create a new ServiceResult

#### DeleteOrderLineByOrderID(int id, out ServiceResult result)

1. Connect to db
2. Run delete query to delete ALL of the order lines where the OrderID equals id
3. Close connection
4. Create a new ServiceResult

#### GetMeals(out ServiceResult result)

1. Create a list of meal objects
2. Connect to db
3. Select everything from the tblMeals table
4. Execute reader
5. Loop through each row and add it to the meal list
6. Close connection
7. Create a new ServiceResult
8. Return meals as an array

#### UpdateMeal(Meal m, out ServiceResult result)

1. Connect to db
2. Run update query with specified data
3. Close connection
4. Create a new ServiceResult

#### DeleteMeal(int id, out ServiceResult result)

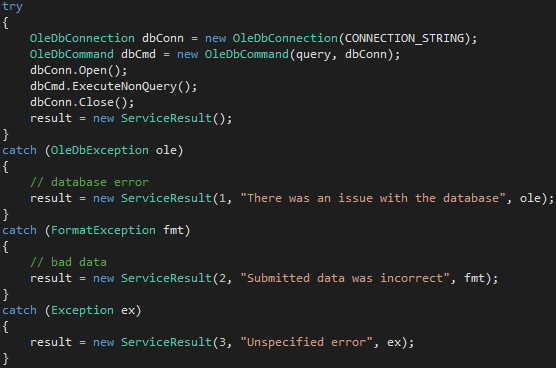
1. Connect to db
2. Run delete query to delete the meal where the ID equals the specified id
3. Close connection
4. Create a new ServiceResult

#### AddMeal(string mealName, out ServiceResult result)

1. Connect to db
2. Run insert query with specified data
3. Close connection
4. Create a new ServiceResult

## Error Handling

Each of the web methods within the service have try-catch with multi-catching. They catch specific exceptions so that the ServiceResult can be given specific information so that the output can be more user friendly.



This works because the ServiceResult class contains an error code, a user-friendly message, an exception type, and an exception message. The error code numbers do not matter, as long as it is consistent, because they simply tell the developers what the issue is, without giving anything away to the user.

# Testing

Testing was done on 13/09/16 by Daniel Field, and Brian Field. The functionality tested was the ordering system, as well as some aspects of the user account system. The grey box testing was useful in that it clearly pointed out the issues with the system, with the most critical issue being XSS attacks.

## Grey Box Test

This test was done by **Brian Field**, with very limited knowledge of the systems inner workings.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Cases** | **Expected Result** | **Actual Result** | **Criticality**  **(Low, Moderate, High, N/A)** | **Pass or Fail** | **Additional Comments** |
| Login Test Cases | | | | | |
| Attempt logging in with no user name and password | Display error “Invalid credentials” | Works OK | N/A |  |  |
| Attempt logging in with “dev” as the user name, and no password | Display error “Invalid credentials” | Worked OK | N/A |  |  |
| Attempt Log in with “dev” and put “password” as the password | Display error “Invalid credentials” | Displayed Invalid Credentials | N/A |  |  |
| Attempt logging in with “dev” as both the username and password | Redirect to Default page. | Logged into Kitchen Web Lobby | N/A |  |  |
| Ordering Test Cases | | | | | |
| Make a new order | Display a new row in the table with the current date/time, and empty delivery address and empty phone number fields | Created Order as required | N/A |  |  |
| Edit the order and put “56 Newman Rd, Madeuptown, QLD 1453” as the address, and “0423 321 414” as the phone number | Display “operation successful” | Edited it to add the address OK | N/A |  |  |
| Delete the newly created order | Remove the row from the table and display “operation successful” | Deleted OK | N/A |  |  |
| Add “Sausage on Bread” with a quantity of 5 to the order. | Display “operation successful” | Added Item OK | N/A |  |  |
| Add “Chicken Sandwich” with a quantity of -5 to the order. | Display “operation successful” | Added -5 OK | High |  | It can add Negative numbers into the field |
| Delete the chicken sandwich item from the order | Remove from the table and display “operation successful” | Doesn’t Refresh on Delete | High |  |  |
| Return to the orders page and select the first order in the table | Redirect to the order editor page and display the order information | Retuned to Orders page. Selected first Order | N/A |  |  |
| Account Creation/Editing Test Cases | | | | | |
| Create a user called “BobSmith63.2&” | Add the new row and display an asterisk before the name | Worked OK | N/A |  |  |
| Create a user called “<script>alert(‘hello, world’)</script>” | Throw an exception | Threw an Exception | High |  |  |
| Change the password of “user2” to “1234567890” | Display “operation successful” | Worked OK | N/A |  |  |
| Change the password of “user2” to “1234”, but put “1234567890” in the confirmation field | Display “Error. Please ensure that the passwords match.” | Passwords did not match | N/A |  |  |
| Change the permissions so that “user2” cannot write to the orders page, but can still read from it. | Display “successful operation” and have the table display “1” as the access level for the order role | Worked OK | N/A |  |  |

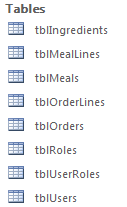
## White Box Test

This test was done by myself (**Daniel Field**), with knowledge of how the system works internally.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Cases** | **Expected Result** | **Actual Result** | **Criticality**  **(Low, Moderate, High, N/A)** | **Pass or Fail** | **Additional Comments** |
| Login Test Cases | | | | | |
| Attempt logging in with no user name and password | Display error “Invalid credentials” | Worked flawlessly | N/A |  |  |
| Attempt logging in with “dev” as the user name, and no password | Display error “Invalid credentials” | Worked flawlessly | N/A |  |  |
| Attempt Log in with “dev” and put “password” as the password | Display error “Invalid credentials” | Worked flawlessly | N/A |  | Did not login because the password was not the default password (‘password’) |
| Attempt logging in with “dev” as both the username and password | Redirect to Default page. | Redirected to the default page correctly | N/A |  |  |
| Ordering Test Cases | | | | | |
| Make a new order | Display a new row in the table with the current date/time, and empty delivery address and empty phone number fields | Works as intended | N/A |  |  |
| Edit the order and put “56 Newman Rd, Madeuptown, QLD 1453” as the address, and “0423 321 414” as the phone number | Display “operation successful” | Works correctly | N/A |  |  |
| Delete the newly created order | Remove the row from the table and display “operation successful” | Working as intended | N/A |  |  |
| Add “Sausage on Bread” with a quantity of 5 to the order. | Display “operation successful” | Added item successfully | N/A |  |  |
| Add “Chicken Sandwich” with a quantity of -5 to the order. | Display “operation successful” | Added item successfully, unfortunately |  |  | Should be an unsigned integer to prevent negative values. |
| Delete the chicken sandwich item from the order | Remove from the table and display “operation successful” | The info label displays “successful operation”, though it did not update the table | High |  | After finding this error, the deletion event has been modified to reload the data from the database. Problem is now solved |
| Return to the orders page and select the first order in the table | Redirect to the order editor page and display the order information | Worked correctly | N/A |  |  |
| Account Creation/Editing Test Cases | | | | | |
| Create a user called “BobSmith63.2&” | Add the new row and display an asterisk before the name | Worked in the way it was expected | N/A |  |  |
| Create a user called “<script>alert(‘hello, world’)</script>” | Throw an exception | Did not work, which was expected. | High |  |  |
| Change the password of “user2” to “1234567890” | Display “operation successful” | Displayed “successful operation” | N/A |  |  |
| Change the password of “user2” to “1234”, but put “1234567890” in the confirmation field | Display “Error. Please ensure that the passwords match.” | Displayed error message | N/A |  |  |
| Change the permissions so that “user2” cannot write to the orders page, but can still read from it. | Display “successful operation” and have the table display “1” as the access level for the order role | Working as expected | N/A |  |  |

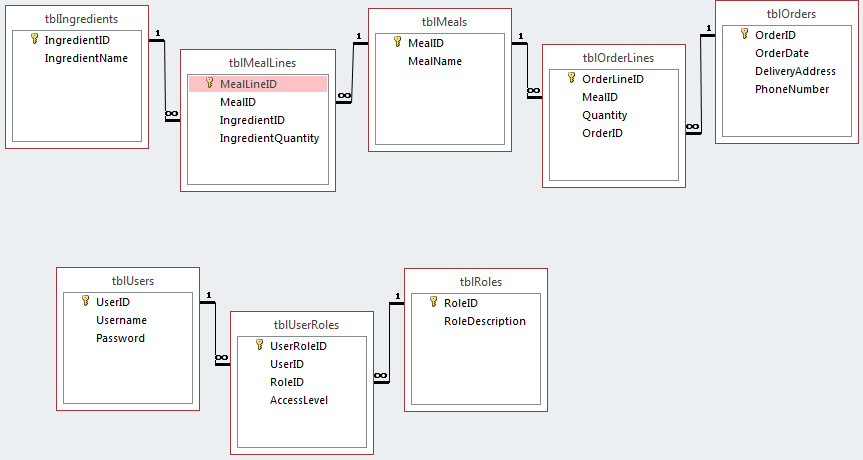
# The Database

The system uses a Microsoft Access relational database located in the [](CKService/App_Data)folder within the web service, and contains the following tables:



## Relationships

Most of the tables in this database are related, however tblUser, tblUserRoles, and tblRoles are connected with each other, but not the rest of the database.



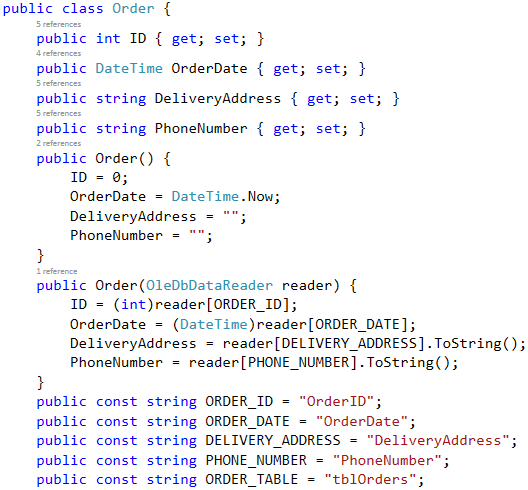
## Attributes

There are many attributes that are missing from the database, that are quite critical in making this system work correctly. One of which is in the meal lines table. This table does not contain any units of measurement, and only contains the quantity. The quantity is useless in most cases if you do not know how you are meant to measure each ingredient. For example, if you select ‘white bread’, there is no way of knowing if that is a whole loaf, or just a single slice.

Before deploying this system, it will be critical that the database is normalised and contains all of the information necessary to be able to replace the old paper-based system that the charity kitchen is currently using.

## Entity-related Classes

Each table in the database has a class that stores single rows of their data. One of these classes is the Orders class which is in the file ‘Orders.cs’. Here is a screenshot of that class.



This class contains four properties. Each property represents an attribute in the database.

There are two constructors, the first constructor is called when it is necessary to have an object with the default property values set. The second constructor takes an OleDbDataReader instance as a parameter, which is used to set the values of the properties.

# Advantages and Limitations of HTTP-based Applications

There are many advantages to having an HTTP-based application. Here are some advantages, and some limitations of using HTTP:

* It is very good for transferring hypertext – It is far better at transferring hypertext than most other protocols. This is the reason HTTP is so common.
* Mapping IP addresses to domain names – This made the internet much more popular by making it easier to access websites.
* It’s simple – HTTP is coded in plain text, making it easy to implement.

Disadvantages:

* It does not encrypt the data – this makes it unsuitable for doing anything secretive, logging into anything, and banking.
* Most webpages require multiple requests and responses to transmit the content.

Why not just use a Windows application instead? This is a question that must always be considered. If you need to access the application on a large variety of computers, it is often a better idea to use a web application. Though, you don’t necessarily have to use a web browser application to use HTTP.

To WinForms, or not to WinForms. That is the question.

# Design Patterns in the Microsoft .NET Framework

Design patterns are ways of solving issues that can commonly occur within the software engineering domain. The Microsoft .NET Framework uses many design patterns that can be seen throughout the computing world.

There are different sub-categories of design patterns: creational patterns, structural patterns, and behavioural patterns. There are 23 standard patterns that are used, these are known as the ‘Gang of Four (GoF) design patterns.

Here are a few of the creational design patterns that are used in the Microsoft .NET Framework:

* Factory Method – A class that returns an object, but gets subclasses to deal with the instantiation of the object.
* Prototype – An instance of a class to be cloned
* Singleton – This is a class which can only have one instance.

Below is a list of a few structural design patterns:

* Composite – This is a tree structure of objects. Examples include linked lists, and binary trees
* Proxy – The proxy design pattern is simply an object that represents another object
* Facade – A class that interfaces to multiple interfaces in a sub-system

Here is a list of a few behavioural design patterns:

* Mediator – This design pattern mediates the communication between multiple classes.
* Interpreter – This design pattern is for defining a representation of a language’s grammar with an interpreter that interprets the language’s syntax. This is not used as often as the previously mentioned patterns
* Iterator – This design pattern used for looping through an array of elements (literal meaning of array, not the programming related meaning) from within an object in a sequential order.

The Microsoft .NET Framework uses these design patterns, and more, to provide ease of use for developers, and to create software that is faster, more space efficient, and cost effective.

Disclaimer: The accuracy of this research depends on the accuracy of the sources of information that were used throughout the writing of this report. Please see the ‘References’ heading for hyperlinks to the sources.

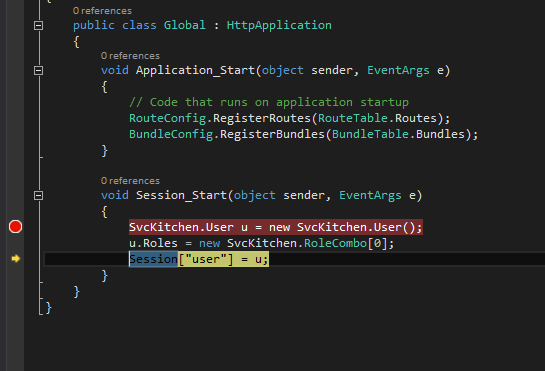
# Debugging Techniques

Many debugging techniques were used throughout the debugging process of this application. Here are three techniques that were used.

## Breakpoints

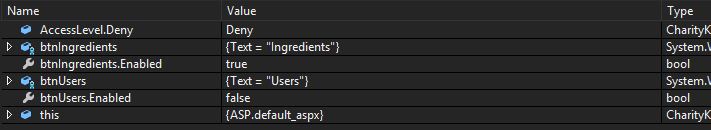
Breakpoints are very useful in debugging an application that is written in managed code. They allow you to step through the code of the application line-by-line, to allow you to see where the code could be having errors.

Breakpoints allow you to use watches, which are explained under the next heading.



## Watches

Watches allow the developer to see the values of specific variables while they are debugging an application. This generally requires a breakpoint to be triggered before watches are useable.



## Error Messages

The application uses the ServiceResult class to get error messages from the web methods. These messages display user-friendly messages that let the user know what they may have done wrong, as well as let the developer know exactly where the error has occurred.

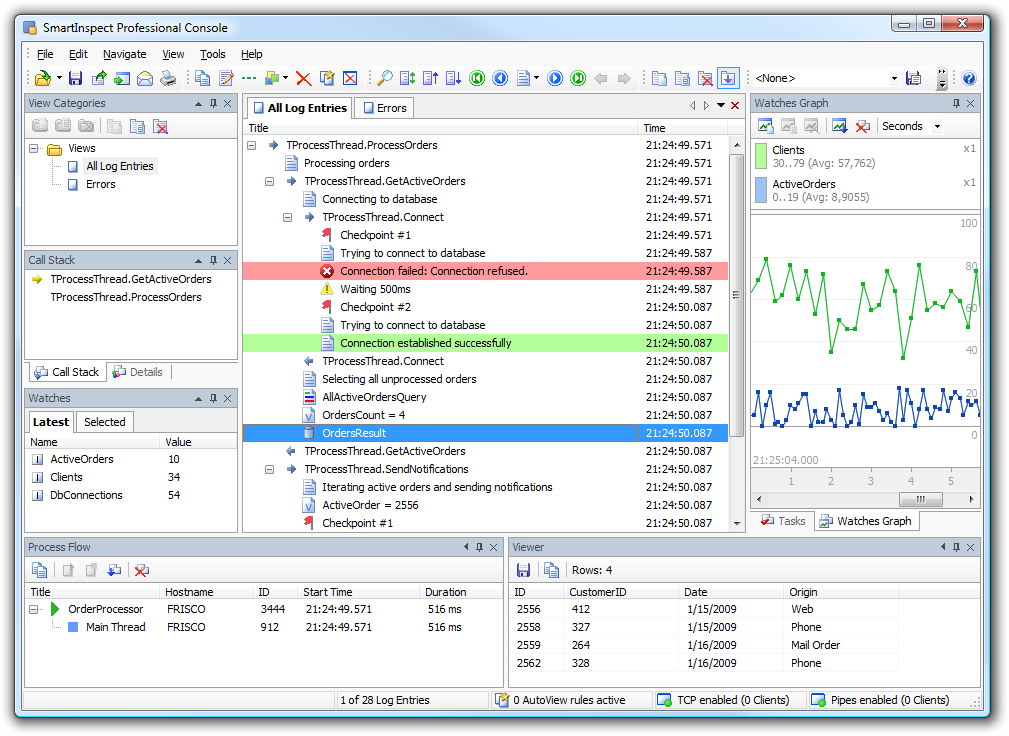
This is explained further under the “Error Handling” section, under the heading “Web Service”. (Screenshot of the error messages included there)

# Three Logging Frameworks Available for .NET

## SmartInspect

SmartInspect is an advanced logging and tracing tool. It allows you to track errors, objects and database results; as well as view an advanced screen for log analysis and monitoring.

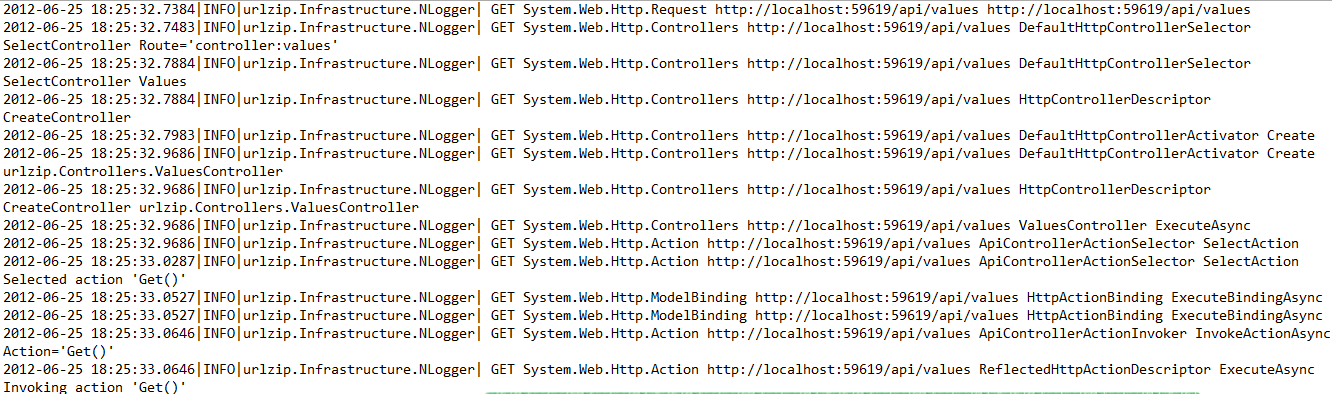
It has the ability to log to binary files or text files, and has encryption capability.



## NLog

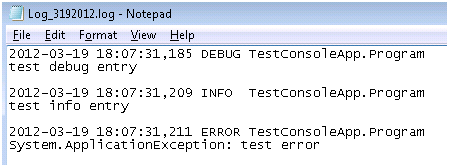
NLog is a free open-source logging platform for .NET, Silverlight, Windows Phone, and Xamarin with the ability to create highly-detailed logs for your application.

NLog has the ability to output log messages to files, databases, consoles, e-mail, and ASP.NET trace.



## Log4net

Apache log4net is a port of the original log4j. It is a tool that logs detailed error information and outputs to a variety of targets, such as text files. This tool is easy to use, and has hierarchical logging architecture, XML configuration, and it works with all of the .NET Framework versions.



# Site Map



Above is a basic navigation map of the system. Many of the pages are designed to do multiple things, as you can see in the ingredients page, the roles page, and the meals page, they can handle all of the functionality on single pages.

The Orders page does not have as much functionality, and is split into an orders page, and an editing page, instead of having panels pop-up.

# References

The advantages of hypertext transfer protocol: <https://www.techwalla.com/articles/the-advantages-of-hypertext-transfer-protocol>

Out (C# Reference): <https://msdn.microsoft.com/en-us/library/t3c3bfhx.aspx>

How to: Create a Web Service Method: <https://msdn.microsoft.com/en-us/library/4ef803zd(v=vs.90).aspx>

.NET Design Patterns: <http://www.dofactory.com/net/design-patterns>

Behavioural, Creational, Structural Types of Design Patterns: <http://www.gofpatterns.com/design-patterns/module2/behavioral-creational-structural.php>

WebReference

WebReferenceRevised

GloriousReference

Comparison of .NET Logging Frameworks and Libraries: <http://www.dotnetlogging.com/comparison/>

SmartInspect: <http://www.gurock.com/smartinspect/>

NLog: <http://www.nlog-project.org/>

Log4net: <http://logging.apache.org/log4net/>

Software used for site map: <https://www.mindmup.com/>