**DESIGN DOCUMENT**

**DEV PROJECT SAMPLE**

**TAREK SELEM**

**21-02-2018**

**Version 0.1**

| REVISION HISTORY | | | |
| --- | --- | --- | --- |
| DATE | VERSION | DESCRIPTION | AUTHOR |
| 21-02-2018 | 0.1 | Initial Version | Tarek Selem |
|  |  |  |  |

Table of Contents

[DEV PROJECT SAMPLE 1](#_Toc507007586)

[TAREK SELEM 1](#_Toc507007587)

[21-02-2018 1](#_Toc507007588)

[Version 0.1 1](#_Toc507007589)

[INTRODUCTION 3](#_Toc507007590)

[PURPOSE 3](#_Toc507007591)

[PROJECT SCOPE 3](#_Toc507007592)

[DESCRIPTION 3](#_Toc507007593)

[FEATURES 3](#_Toc507007594)

[OPERATING ENVIRONMENT 3](#_Toc507007595)

[CONTRAINTS: IMPLEMENTATION / DESIGN 3](#_Toc507007596)

[TECHNOLOGY STACK 4](#_Toc507007597)

[BACKEND ARCHITECURE 5](#_Toc507007598)

[SYSTEM FEATURES 7](#_Toc507007599)

[DATA MIGRATION MODULE 7](#_Toc507007600)

[CUSTOMERS MODULE 7](#_Toc507007601)

[ORDERS MODULE 7](#_Toc507007602)

# INTRODUCTION

## PURPOSE

This a sample development task represents the implemented architecture, technical stack, business features and covered areas with automated unit tests.

## PROJECT SCOPE

I tried hard to represent my development skills in this project within a short time.

Project has minor CRUD operations features for a small relational database.

# DESCRIPTION

## FEATURES

System has minor features for two main entities (Customer and Order) with full three modules:

* XML Data migration.
* Customers
* Orders

## OPERATING ENVIRONMENT

1. **BACK END**:
2. Database : Microsoft SQL Server
3. Web API: IIS
4. **FRONT END**: Any exposed domain as it is (HTML, JavaScript) files.

## CONTRAINTS: IMPLEMENTATION / DESIGN

1. Implemented architecture depends on MS SQL Server.
2. For any further relational database provider just implement interfaces (IRepository, IUnitOfWork) which are in assembly name “Futura.DataAccess.Common”.
3. The implemented library (Entity Framework) supports lazy loading data including, so, you need to include the needed relational entity as a parameter in repository.
4. For any NoSQL or Block Chain databases providers add new needed interfaces in assembly “Futura.DataAccess.Common”.
5. Setting the database connection string from the application configurations file.
6. Use Entity Framework CLI for creating the database as it implemented Code First approach.
7. Web API will log the errors in path “{basedir}\ApiLog\{date}-webapi.log”.
8. XML data migration file for import data has to be formatted like the following:

<? xml version=""1.0"" encoding=""utf-8""?>

<Root>

<Customers>

<Customer CustomerID="GREAL">

<CompanyName>Great Lakes Food Market</CompanyName>

<ContactName>Howard Snyder</ContactName>

<ContactTitle>Marketing Manager</ContactTitle>

<Phone>(503) 555-7555</Phone>

<FullAddress>

<Address>2732 Baker Blvd.</Address>

<City>Eugene</City>

<Region>OR</Region>

<PostalCode>97403</PostalCode>

<Country>USA</Country>

</FullAddress>

</Customer>

<Customers>

<Orders>

<Order>

<CustomerID>LETSS</CustomerID>

<EmployeeID>4</EmployeeID>

<OrderDate>1998-02-12T00:00:00</OrderDate>

<RequiredDate>1998-03-12T00:00:00</RequiredDate>

<ShipInfo ShippedDate="1998-02-13T00:00:00">

<ShipVia>2</ShipVia>

<Freight>90.97</Freight>

<ShipName>Let's Stop N Shop</ShipName>

<ShipAddress>87 Polk St. Suite 5</ShipAddress>

<ShipCity>San Francisco</ShipCity>

<ShipRegion>CA</ShipRegion>

<ShipPostalCode>94117</ShipPostalCode>

<ShipCountry>USA</ShipCountry>

</ShipInfo>

</Order>

</Orders>

</Root>

## TECHNOLOGY STACK

System implemented as Single Page Application architecture

**BACK END**:

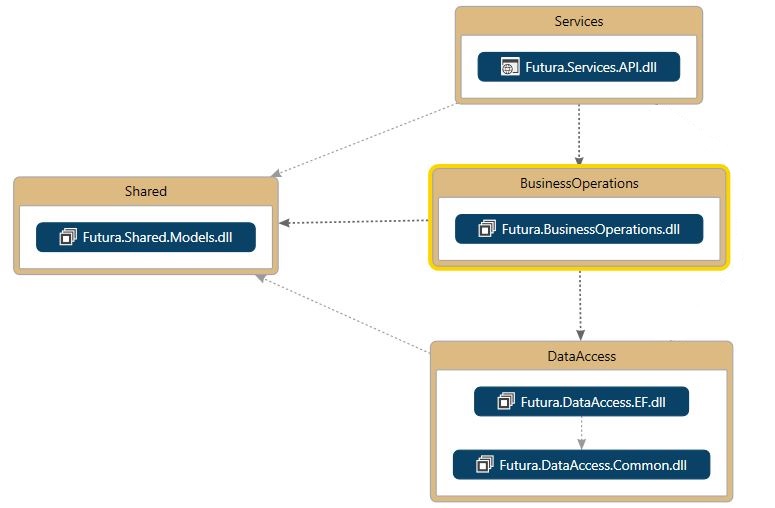
1. Microsoft SQL Server Database
2. .NET Framework (C#)
3. REST ASP.NET MVC Web API 2
4. Entity Framework
5. AutoMapper
6. Ninject
7. NUnit
8. Moq
9. NLog

**FRONT END**:

1. HTML
2. JavaScript Framework Angular4
3. Bootstrap

## BACKEND ARCHITECURE

The backend contains five main areas:

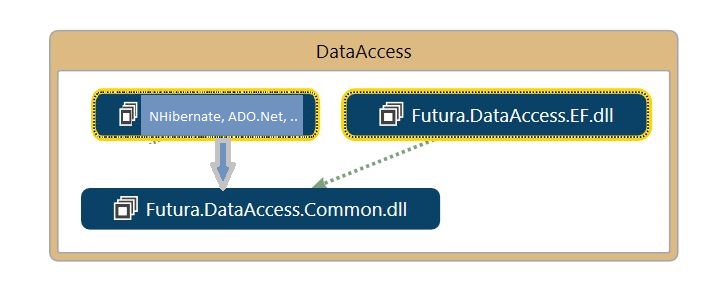


1. **Shared:**

Here we add any shared libraries across the solution like View models, Binding models or Entities.

1. **Data Access:**

Contains the common interfaces that the higher layers communicate with and the data access libraries like (Entity Framework, NHibernate, ADO.NET, etc.) that implement the common interfaces which implement Unit of work and repository patterns.



1. **Business Operations:**

Contains a manager class for each module which implements an interface of all the module operations rather than models mapping configurations.

1. **Services/Presentation Layer:**

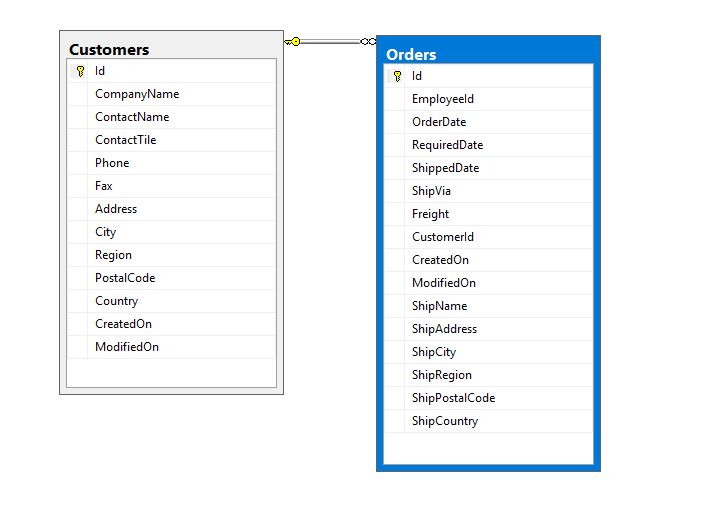
This area contains a RESTful web API which represents the business operations to be exposed for any client consumer like Web Portal, Mobile Application, etc.

1. **Tests:**

Contains all automated test cases for the application layers to keep on our code base healthy, clean and having more confidence while the refactoring processes.

Currently only the Business Operations layer is covered with automated test and because of the limited provided time for the task the data access layer and web API controllers weren’t covered. Nonetheless they have to be covered with automated test cases in real projects.

Database tables diagram:



# SYSTEM FEATURES

## DATA MIGRATION MODULE

|  |  |
| --- | --- |
| **DESCRIPTION** | This module has data migration feature by uploading XML file format with a certain schema as it mentioned [above](#_CONTRAINTS:_IMPLEMENTATION_/). |
| **FUNCTIONAL REQUIREMENTS** | Import XML data file to the application. |

## CUSTOMERS MODULE

|  |  |
| --- | --- |
| **DESCRIPTION** | This module contains customers list, add, edit and delete a customer in addition to the edit screen has customer address details and a list of customer orders. |
| **FUNCTIONAL REQUIREMENTS** | 1. Customers List 2. Customer Edit/Details 3. Add new Customer 4. Delete Customer 5. List Customer Orders |

## ORDERS MODULE

|  |  |
| --- | --- |
| **DESCRIPTION** | This module contains Orders list, add, edit and delete an Order in addition to the edit screen has shipping address details. |
| **FUNCTIONAL REQUIREMENTS** | 1. Orders List 2. Order Edit/Details 3. Add new Order 4. Delete Order |