Intern Program: Week 6 Assignment

**Purpose:** This document outlines the assignment that will be handed out for week 6 of the intern program. This project is to be worked on while you complete the assigned readings for Week 6, and must be completed for your Week 6 review.

# Week 6 Requirements:

This week will require users to build the following elements:

* **Unit Testing:** You will be creating a unit test project to ensure that code changes do not affect existing functionality.
* **Inventory Web Service (StoreFront.InventoryService):** An external application of the inventory management department needs product information from the database. You will be creating a web service to provide remote access.
* **Shipping Department Web API (StoreFront.ShippingApi)**: An external application of the shipping department needs order information from the database so that it can update orders that have shipped. You will be creating a web API to provide shipping with the interface they need.
* **Security:** You will be providing a custom class for managing the security within the application.
* **GitHub:** Make sure to commit your changes to the repository when you are done.

# Technical Requirements:

The following are the technical requirements for each page in this assignment: Below is a listing of the approaches that are to be used to satisfy the business requirements above.

## Unit Testing

The following are the requirements for this feature:

* You will create a new Unit Test project called “StoreFront.UnitTest”.
* You will use the “Arrange-Act-Assert” pattern to create a series of unit tests that test the data layers abilities to Create, Read, Update, and Delete data from the database.

## Inventory Web Service:

The following are the requirements for this feature:

* You will create a new WCF project called “StoreFront.InventoryService”.
* The service should utilize the following methods:
  + SearchProducts(string text): This should search through all existing products that are published in the application and return a List of those objects that either the name or the description contain the search text. This should reference the “InventoryRepository” class in StoreFront.Data.
  + GetProductDetails(int id): This should return all details about a specific product, including potential variants to the requester. It will retrieve this by using the ProductID, which will be passed to it. This should reference the “InventoryRepository” class in the StoreFront.Data.
* You should create a web page within StoreFront that tests communication to the service. This page would normally not be part of the application, and does not need to be styled or linked to.

## Shipping Department Web API:

The following are the requirements for this feature:

* You will create a new ASP.NET project using WebAPI called “StoreFront.ShippingApi”.
* The API should have only 1 controller, “OrderController”
* The Shipping API should utilize the following actions:
  + GetOrders(DateTime startDate, DateTime endDate): This should return all order objects that have an “OrderDate” between the start date and end date. This should use the OrderRepository class to retrieve details about the order the from the StoreFront.Data library.
  + MarkOrderShipped(int id): This should change the status of an order to “Shipped”. This should use the OrderRepository class to update the details about the order from the StoreFront.Data library.
* You should create a unit test which will create an OrderController to test the GetOrders and MarkOrderShipped functionality using the “Arrange-Act-Assert” pattern.
* Furthermore, you must ensure that the api can be called successfully from outside the unit test project, this may be accomplished via a web browser, or plug in such as “Postman”.

## Security:

The following are the requirements for implementing security:

* This application will utilize a new class called “SqlSecurityManager”, and will utilize the information with the User table.
* The application should use forms authentication as outlined in your book, and should be configured using the web.config.
* The “SqlSecurityManager” class will utilize the following methods:
  + AuthenticateUser(string username, string password): This will return a true / false, depending on whether or not the user and password are valid. Also, if so, the username should be added to Session with a key of “UserName”.
  + IsAdmin(): This method will utilize entity framework to check if the username stored in session is an administrator and return a boolean value.
  + LoadUser(username)
    - This method should load the properties of this object based on the username that is given.
  + SaveUser()
    - Updates the database based on the UserID to keep information in sync between the membership provider and the User tables.
  + RegisterUser()
    - This should use the entity framework tables to create the user account in the appropriate table. When users are created they should not be given the admin role.
  + DeleteUser(username)
    - This should use the entity framework tables to remove a user from the database.
* Additionally, your User table should be added to entity framework, all operations in the SqlSecurityManager should use entity framework for the data operations in a new class called “UserRepository”.
* The Login / Register screens should be updated to use the SqlSecurityManager to login or create users.
* The CustomerAdminDetails screen should use the SqlSecurityManager to save all details related to the user.

## GitHub

Ensure that all your changes are committed to your GitHub repository for review.