Online Supermarket – TDD Case Study

An ASP.NET MVC web based application is needed to allow shoppers to shop online for specific grocery items. The application’s features are described below.

# Admin CRUD Features

## Create Item: /item/create/

## Read Item: /item/details/*<item id>*

## Update Item: /item/edit/*<item id>*

## Delete Item: /item/delete/*<item id>*

There must be a built in user in an ‘admin’ role. This user must have email ‘admin@user.com’ and password ‘Pass123!’ Only the admin user is allowed to CRUD supermarket items. When the admin user logs in, he or she is redirected to the items index page (/items/index/). The admin user has no supermarket profile (see below) to maintain. You are allowed to scaffold the controller and views to CRUD the items.

Each item has the following properties:

* **Id** – the item’s id, primary key, auto-increment
* **Name** – the item’s name, required, maximum length: 50
* **Type** – the item’s type, one of: Cereal, Can Goods, Bread, Candy
* **Price** – the item’s price, greater than zero
* **Amount in stock** – must be at least 0 and at most 1000

# Title Bar Feature

The title bar is visible on all pages.

* A nice image (you decide on something appropriate)
* Name of the application: **Online Supermarket**
* ‘Home’ link – takes the user to the Home page
* ‘View Profile’ link – takes the shopper to the view profile page (not visible to admin users)
* ‘About’ link – takes the user to the about page – the about page simply states the name and email of the developer – the email is a mailto link

# Main Shopping

/home/index/

This action redirects admin users to the CRUD items index page.

If a shopper has no supermarket profile, he or she is redirected to create his or her supermarket profile.

The shopper is presented with:

* 4 ‘aisles’
  + Cereal, Can Goods, Bread, Candy
  + Each aisle lists the associated items with the following: name, price, amount in stock, and a **Buy** button. If the amount in stock is zero, then the Buy button is disabled.
* Number of items in shopping cart and the shopping cart’s total. This is displayed near the top and bottom of the view. Note that if the user logs out and then logs back in then the shopping cart data is kept.
* A **View Shopping Cart** link

# Buy Item

/shoppingcart/buy/***<item id>***

Admin users cannot access this action.

This action displays the details of the item and allows the user to enter the quantity to buy. The quantity to buy cannot exceed the item’s amount in stock. When the shopper submits this data, the item is added to the shopping cart, and the amount in stock is reduced by the quantity to buy. The shopper is redirected to the main shopping page after purchase.

There should be a link back to main shopping.

# Shopping Cart

/shoppingcart/index/

Admin users cannot access this action.

This action lists all items in the shopping cart along with the total price for each item bought. There is to be a ‘**Remove**’ link next to each item. There should also be a grand total price of the items in the shopping cart.

There should be 3 links: **Empty cart (No Purchase)**, **Checkout**, and Back to main shopping

# Remove Item

/shoppingcart/remove/***<item id>***

Admin users cannot access this action.

This action displays the details of the item and allows the user to confirm removal. If the shopper confirms removal, the item is removed from the shopping cart, and the amount in stock increased by the quantity to buy. The shopper is redirected to the shopping cart page after removal.

There should be a link back to the shopping cart page.

# Empty Cart

/shoppingcart/empty/

Admin users cannot access this action.

This action allows the user to confirm emptying the cart. If the shopper confirms emptying, the items are removed from the shopping cart, and the amount in stock for each item is increased by the quantity to buy for that item. The shopper is redirected to the main shopping page.

There should be a link back to the shopping cart page.

# Checkout

/shoppingcart/checkout/

Admin users cannot access this action.

This action displays all items and the grand total price for all items in the cart and allows the user to confirm paying for the items in the cart. If the shopper confirms paying, the items are removed from the shopping cart. The shopper is redirected to the main shopping page.

There should be a link back to the shopping cart page.

# Create Profile

/profile/create/

Admin users cannot access this action.

This action allows a shopper to create his or her supermarket profile. The shopper is not allowed back here once profile is created.

* **Email** – Primary key, maps to the identity email (but not a foreign key)
* **Username** – Unique to each shopper, required, does not change once created, not the same as the username in the identity
* **First name** – required, maximum length 50
* **Last name** – required, maximum length 50
* **Street address** – required
* **State** – required, must be two characters, and must be one of the standard US state codes
* **Zip** – required, must be 5 digits

The shopper is redirected to main shopping after the profile is created.

# View Profile

/profile/details/

Admin users cannot access this action.

This action allows the shopper to view his or her supermarket profile:

* Full name (Last, First)
* Street address
* State
* Zip

There must be links to **edit profile**, **delete profile**, and back to main shopping

# Edit Profile

/profile/edit/

Admin users cannot access this action.

This action allows the shopper to edit his or her supermarket profile:

* **First name** – required, maximum length 50
* **Last name** – required, maximum length 50
* **Street address** – required
* **State** – required, must be two characters, and must be one of the standard US state codes
* **Zip** – required, must be 5 digits

There must be a link back to main shopping.

The shopper is redirected to view profile after completion.

# Delete Profile

/profile/delete/

Admin users cannot access this action.

This action allows the shopper to delete his or her profile after confirmation.

There must be a link back to main shopping.

The shopper is redirected to main shopping after completion.

# Application Development using TDD

## Admin CRUD Features

|  |
| --- |
| Feature: AdminUser  In order to keep the online supermarket running  As an Admin User  I want to CRUD the items in the online supermarket  @loggingIn  Scenario: I log in  Given I am on the login page  And I have entered admin@user.com as the username  And I have entered Pass123! as the password  When I issue the login command  Then the items index page is displayed |

* Install Specflow.NUnit
* Default step generation: <trace stepDefinitionSkeletonStyle="MethodNamePascalCase" />
* Generate the step definitions: class name = ILogIn
* Modified the generated methods to accurately identify the parameters:

|  |
| --- |
| [Given]  public void GivenIHaveEntered\_USERNAME\_AsTheUsername(string username)  {  ScenarioContext.Current.Pending();  }  [Given]  public void GivenIHaveEntered\_PASSWORD\_AsThePassword(string password)  {  ScenarioContext.Current.Pending();  } |

* Code the first step:

|  |
| --- |
| [Given]  public void GivenIAmOnTheLoginPage()  {  OnlineSupermarketApp.Initialize();  OnlineSupermarketApp.LoginPage.Goto();  } |

* Create new class library project: OnlineSupermarket.TestingFramework
* Establish the project reference to OnlineSupermarket.TestingFramework
* Create the public static class OnlineSupermarketApp
* Generate method stub for Initialize()
* Create inner class: public static class LoginPage
* Generate method stub for Goto()
* Code Initialize() and LoginPage.Goto()

|  |
| --- |
| private const string HostUrl = "http://localhost:23123/";  private const string LoginUrl = HostUrl + "Account/Login/";  public static void Initialize()  {  Chrome.Initialize();  }  public static class LoginPage  {  public static void Goto()  {  Chrome.Goto(LoginUrl);  }  } |

* Create new class library project: OnlineSupermarket.WebDriverFramework
* Set the project reference to OnlineSupermarket.WebDriverFramework
* Create public static class Chrome
* Generate the method stubs for Chrome.Initialize() and Chrome.Goto()
* Download ChromeDriver – WebDriver
* Install Selenium WebDriver and Selenium.Support
* Add the web driver code to Chrome:

|  |
| --- |
| public static class Chrome  {  private static IWebDriver \_page = null;  public static void Initialize()  {  \_page = new ChromeDriver(@"Q:\");  }  public static void Goto(string url)  {  \_page.Navigate().GoToUrl(url);  }  } |

* Running the test produces a message that the site cannot be reached.
* Write code to quit the testing:

|  |
| --- |
| [Binding]  public class ILogIn  {  [Before]  public static void Setup()  {  OnlineSupermarketApp.Initialize();  }  [After]  public static void TearDown()  {  OnlineSupermarketApp.EndTest();  }  [Given]  public void GivenIAmOnTheLoginPage()  {  OnlineSupermarketApp.LoginPage.Goto();  } |
| public static class OnlineSupermarketApp  {  …  public static void EndTest()  {  Chrome.Quit();  } |
| public static class Chrome  {  …  public static void Quit()  {  Thread.Sleep(2000);  \_page.Dispose();  \_page.Quit();  }  } |

## Creating the Initial Application

* Create ASP.NET MVC web application: OnlineSupermarketApp with authentication set to Individual User Accounts and then set it as the startup project.
* Examine the project properties, take note of the project Url, and then modify the testing framework accordingly
* Ensure that the application can remain executing if you stop debugging
* Create a folder and name it DataContexts
* To the DataContexts folder, create a class called IdentityDb
* In the Models folder, open the IdentityModels.cs file, and then move the ApplicationDbContext class to IdentityDb. Overwrite IdentityDb with ApplicationDbContext and then rename it to IdentityDb.
* Build the application to identify the errors that were created by the rename and then fix the errors.
* Enable migrations by typing the following command in the package manager console: enable-migrations -ContextTypeName IdentityDb -MigrationsDirectory DataContexts\IdentityMigrations
* Add explicit migrations by typing the following command into the package manager console: add-migration -ConfigurationTypeName OnlineSupermarketApp.DataContexts.IdentityMigrations.Configuration "InitialCreate"
* Update the root Web.Config to change the name of the mdf file and the database:

|  |
| --- |
| <connectionStrings>  <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\**OnlineSupermarketApp.mdf**;Initial Catalog=**OnlineSupermarketAppDb**;Integrated Security=True"  providerName="System.Data.SqlClient" />  </connectionStrings> |

* Update the database by typing the following command in the package manager console: update-database -ConfigurationTypeName OnlineSupermarketApp.DataContexts.IdentityMigrations.Configuration -verbose
* Run the application and then stop debugging
* You should now be able to run the test case and see something happen

## Code the Username step

* Code the ILogIn username step

|  |
| --- |
| [Given]  public void GivenIHaveEntered\_USERNAME\_AsTheUsername(string username)  {  OnlineSupermarketApp.LoginPage.Username = username;  } |

* Code the LoginPage.Username setter:

|  |
| --- |
| public static string Username  {  set { Chrome.Username = value; }  } |

* Code the Chrome.Username setter:

|  |
| --- |
| public static string Username  {  set  {  var usernameElement = \_page.FindElement(By.Id("Email"));  usernameElement.Clear();  usernameElement.SendKeys(value);  }  } |

## Code the Password step

* Code the ILogIn step

|  |
| --- |
| [Given]  public void GivenIHaveEntered\_PASSWORD\_AsThePassword(string password)  {  OnlineSupermarketApp.LoginPage.Password = password;  } |

* Code the LoginPage.Password setter

|  |
| --- |
| public static string Password  {  set { Chrome.Password = value; }  } |

* Code the Chrome.Password setter and then refactor:

|  |
| --- |
| public static string Username  {  set  {  Chrome.SendKeysById("Email", value);  }  }  public static string Password  {  set  {  Chrome.SendKeysById("Password", value);  }  }  private static void SendKeysById(string elementId, string value)  {  var usernameElement = \_page.FindElement(By.Id(elementId));  usernameElement.Clear();  usernameElement.SendKeys(value);  } |

## Code the Login Command Step

* Code the ILogIn step

|  |
| --- |
| [When]  public void WhenIIssueTheLoginCommand()  {  OnlineSupermarketApp.LoginPage.IssueLoginCommand();  } |

* Code LoginPage.IssueLoginCommand()

|  |
| --- |
| public static void IssueLoginCommand()  {  Chrome.ClickLoginButton();  } |

* Code Chrome.ClickLoginButon()

|  |
| --- |
| public static void ClickLoginButton()  {  \_page.FindElement(  By.CssSelector(@"#loginForm > form > div:nth-child(7) > div > input"))  .Click();  } |

## Code the Items Index Page Assert

* Code the ILogIn step

|  |
| --- |
| [Then]  public void ThenTheItemsIndexPageIsDisplayed()  {  var itemPageTitle = OnlineSupermarketApp.ItemPage.Title;  Assert.That(itemPageTitle, Is.EqualTo("Online Supermarket Items"));  } |

* Code ItemPage.Title

|  |
| --- |
| public static class ItemPage  {  public static string Title  {  get  {  return Chrome.PageTitle;  }  }  } |

* Code Chrome.PageTitle

|  |
| --- |
| public static string PageTitle  {  get  {  return \_page.Title;  }  } |

## Add the Admin User

* In the Identity Configuration, update the Seed method to add the admin user

|  |
| --- |
| protected override void Seed(OnlineSupermarketApp.DataContexts.IdentityDb context)  {  var roleStore = new RoleStore<IdentityRole>(context);  var roleManager = new RoleManager<IdentityRole>(roleStore);  var userStore = new UserStore<ApplicationUser>(context);  var userManager = new UserManager<ApplicationUser>(userStore);  if (!context.Users.Any(u => u.UserName == "admin@user.com"))  {  var user = new ApplicationUser  {  Email = "admin@user.com",  EmailConfirmed = true,  UserName = "admin@user.com"  };  userManager.Create(user, "Pass123!");  roleManager.Create(new IdentityRole { Name = "admin" });  userManager.AddToRole(user.Id, "admin");  }  } |

* Build the project and then update the database
* Update the Home Index action method:

|  |
| --- |
| [Authorize]  public class HomeController : Controller  {  public ActionResult Index()  {  if (User.IsInRole("admin"))  {  return RedirectToAction("Index", "Item");  }  return View();  } |

* Code the Item Index action method, view, and update \_Layout:

|  |
| --- |
| [Authorize(Roles ="admin")]  public class ItemController : Controller  {  // GET: Item  public ActionResult Index()  {  return View();  }  } |
| @{  ViewBag.Title = "Online Supermarket Items";  }  <h2>@ViewBag.Title</h2> |
| <title>**@ViewBag.Title**</title> |

* The test should now pass.
* xxx