**CIVICA INDIA - Unit Testing Contest for Vadodara Developers**

Date: 23rd September 2021 Time: 10 AM to 1 PM IST (3 hours)

**PROACT CODE:** Project: Vadodara Practice Initiatives (G236JU)

Activity: Unit Testing Contest

**Local Environment Prerequisites:**

1. Visual Studio 2019 (Professional) with following workloads:

|  |  |
| --- | --- |
| **Category** | **Workload** |
| Web & Cloud | ASP.Net and web development |
| Desktop & Mobile | .Net desktop development |
| Other toolsets | .Net cross-platform development |

1. .Net 5.0 Framework (Microsoft)
2. Git

**XUNIT as Unit Testing Framework:**

To get ready with skills:

* + Pluralsight courses
  + MSDN
  + Online tutorials

Reference links:

<https://anarsolutions.com/automated-unit-testing-tools-comparison/>

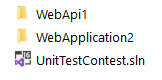
<https://www.lambdatest.com/blog/nunit-vs-xunit-vs-mstest/>

Few recommended courses from Pluralsight:

|  |  |
| --- | --- |
| **Course** | **Link** |
| Unit testing - C# | [https://app.pluralsight.com/library/courses/basic-unit-testing-csharp-developers](https://app.pluralsight.com/library/courses/aspnet-core-fundamentals) |
| Unit testing - xUnit (Getting started) | <https://app.pluralsight.com/library/courses/dotnet-core-testing-code-xunit-dotnet-getting-started> |
| Unit testing - .NET Unit tests with AutoFixture | <https://app.pluralsight.com/library/courses/autofixture-dotnet-unit-test-get-started> |
| Unit testing - Mocking unit tests with Moq | <https://app.pluralsight.com/library/courses/mocking-moq-xunit> |

**Code Folders:**

Once you download code from repos, folder will look like:

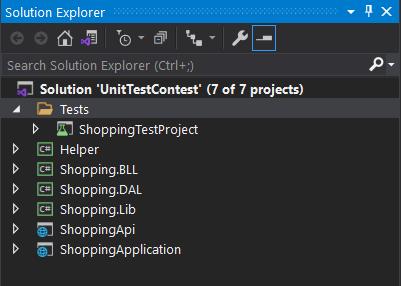


If you want to individually open code for API, there is solution under WebApi1 folder - ShoppingWebApi.sln

If you want to individually open code for MVC application, there is solution under WebApplication2 folder - ShoppingApplication.sln

You will find test project to write unit test cases under folder: WebApplication2\UnitTest\TestProject1

When you will open “UnitTestContest” in VS 2019, solution explorer will look like:



|  |  |
| --- | --- |
| **Project Name** | **Description** |
| ShoppingTestProject | Project to write Unit Test Cases. You can add more test projects with proper naming conventions for different projects. |
| Helper | Utility or helper classes (reusable or common code, which can be shared across multiple projects) |
| Shopping.BLL | Business Logic |
| Shopping.DAL | Data Access Layer. No real database is used in the application. However, for live projects, this project will be interacting to live database. |
| Shopping.Lib | Model Class Library |
| ShoppingApi | Web API for application |
| ShoppingApplication | ASP.Net Core based MVC Application |

For simplicity, interfaces and classes are not separated into different folders.

**Web API details:**

Project: ShoppingApi

**GET** **api/product/GetAll**– Lists all the products.

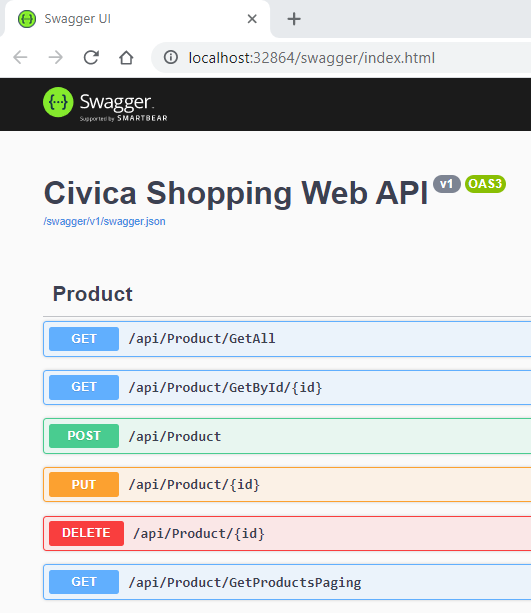
**GET api/product/GetById/{id}**– Gets the details of a product by id.

**POST api/product/Create** – Creates a new product.

**PUT api/product/Update/{id}**– Edits product by id.

**DELETE api/product/Delete/{id}** – Deletes the product by id.

**GET api/product/GetProductsPaging** – Gets the list of products by page number and page size.



**Web Application Details:**

Project: ShoppingApplication

This is a very simple Web Application built upon ASP.Net Core MVC, Civica Shopping Web Application. You should keep running both projects:

1. ShoppingApi (default port: 32864)
2. ShoppingApplication (default port: 44341)

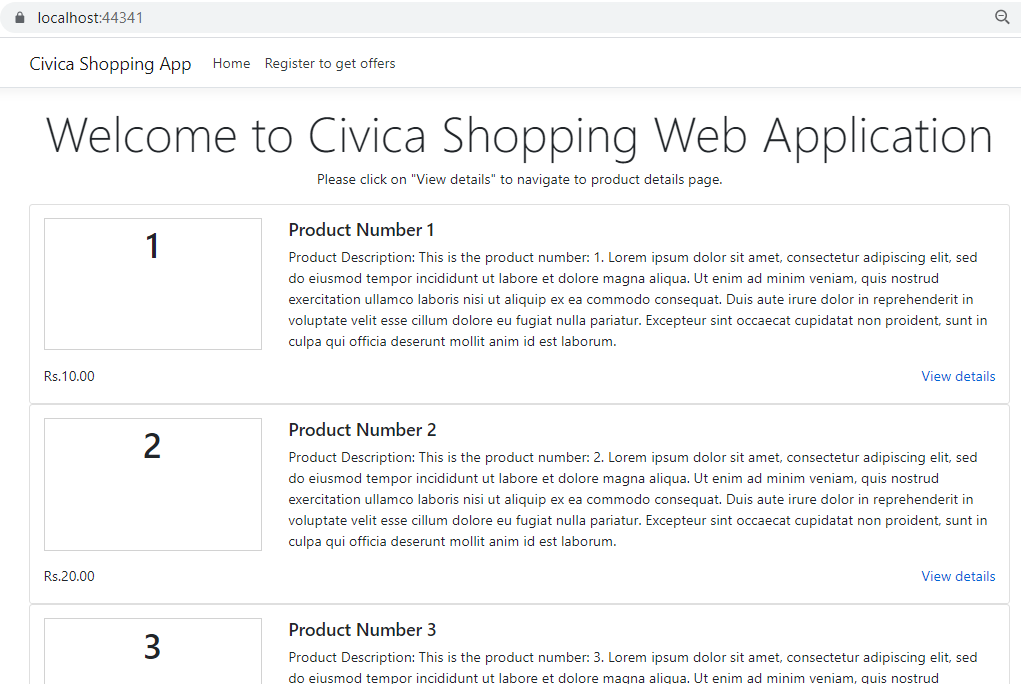
It has only 2 menu items and hence 2 workflows:

1. Home – Lists all products. Clicking on “View details” will navigate to Product details page.
2. Register to get offers – User can fill in personal details to get offers and promo codes etc.

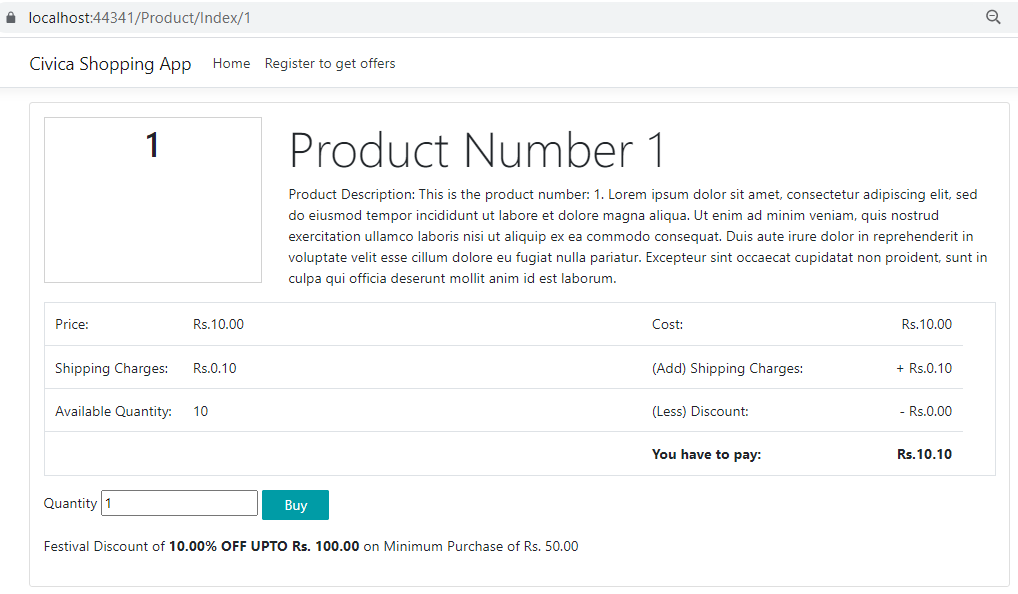
**Workflow of application:**

1. **Product Workflow:**

Home Page:



Clicking on “View details” will navigate to Product Details Page:



User can change quantity and then click on “Buy” button.

Validations on “Buy” buttons will apply for Quantity field (in code, name of property is “PurchasedQuantity”):

1. Quantity should not be blank
2. Quantity should be positive number only
3. Quantity should be not more than available quantity of product

Upon successful validation, code will calculate amounts and will display confirmation screen. While calculating amounts, it will apply discounts if applicable.

Rules for displaying and applying discount:

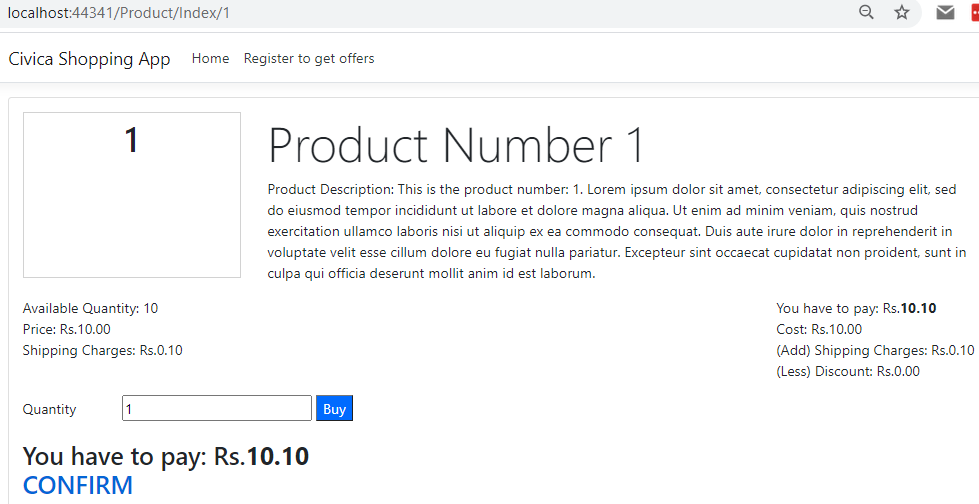
1. Only if merchant has allowed to offer discounts on the product (in code, property with name “IsMerchantDiscountAllowed” will act as flag)
2. Festival Discount – if purchase date (current date) belongs to any festival as per festival repository
3. Promo Code Discount – if user has valid Promo Code

Q: What does “Discount” mean?

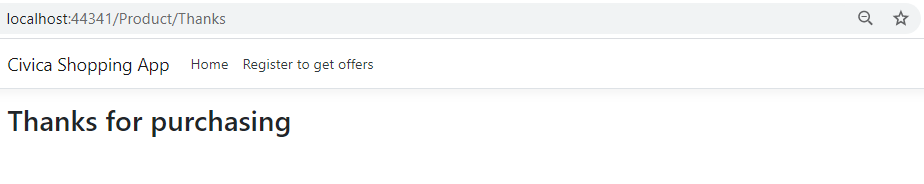
A: Please look into the code for “Discount” class, which is self-explanatory.

To keep UI simple, Promo Code Discount is not implemented on UI layer. However, business logic is written into “Shopping.BAL” project and you need to write Unit Tests for them.

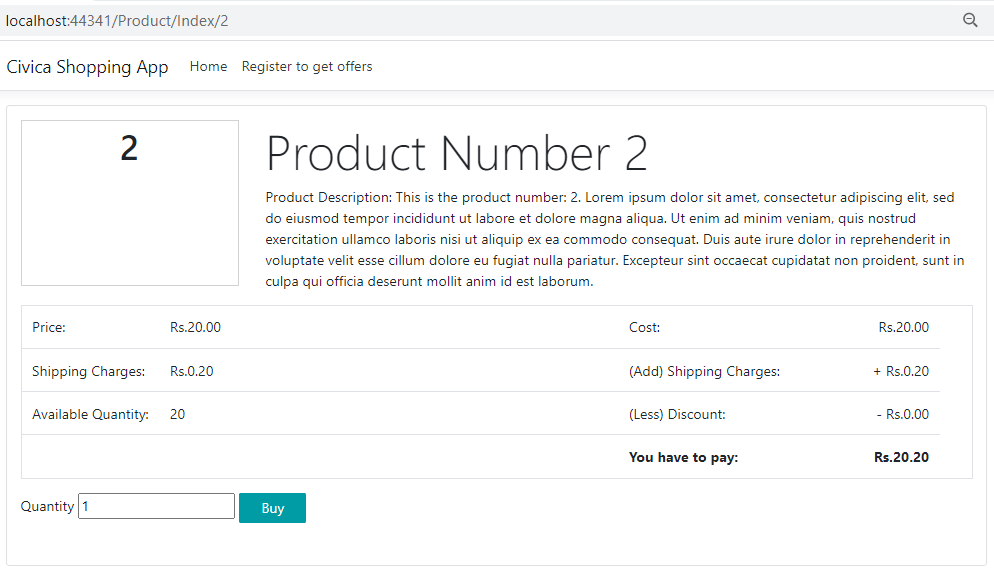
Confirmation screen will look like:



Clicking on confirm, it will navigate to Thanks Page and workflow will end.

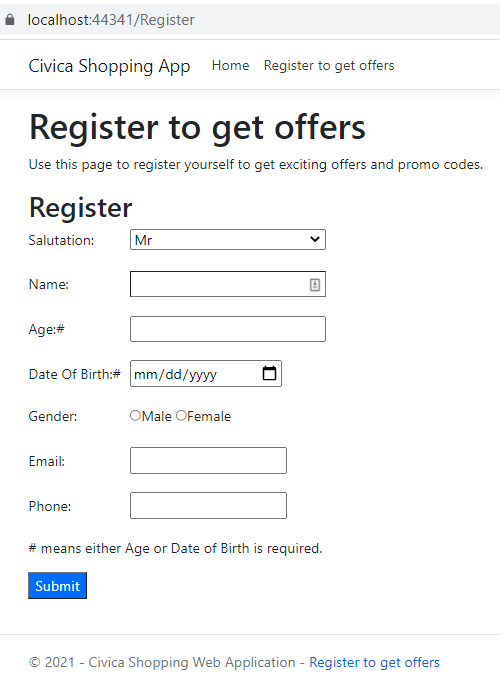


If discount is not applicable on the product, you will be not able to see discount specific line on UI.



1. **Registration Workflow:**

User can navigate to registration page by either clicking on menu item or footer link, named “Register to get offers”. It is very simple form to capture user details. Please look into code to know more details. You have to write as much as possible Unit Tests.



Click on “Submit” button will navigate to “Thanks” page.

