# Orbus Developer Test

The following exercise will test your core development skills. This will help us gauge where your strengths lie and see if you are fit for the role. Tasks should be completed in numerical order, and you should try and follow broadly accepted coding and layout standards when implementing the solution.

Where applicable, instructions and assistance can be found in the file you are instructed to modify in the form of a TODO comment.

The Test covers data retrieval, MVC and JavaScript skills, and is as follows. Best of luck:

1. Complete implementation of the **ProductRepository** class (check TODOs)
2. Complete implementation of the **CategoryRepository** class (check TODOs)
3. Modify the **ProductController** class and use **ProductRepository** to implement all the methods within it.
4. Creating Views

At this point you have the choice to use a Razor/JQuery or Angular approach to completing the rest of the test. If you are comfortable with Angular, use the OrbusDevTest.Angular Project and set this as the StartUp project, otherwise use the OrbusDevTest Project.

**For an Angular Implementation only:**

You will need to use app.js (Scripts/app/app.js) to implement the controllers, routes and logic for the following steps.

* 1. Partials/product.cshtml
     1. This should list Products and include Edit and Details links for each row (these pages will be implemented later)
     2. The input controls are using a data-driven input directive (dynamicInput.js). Have a look at the dynamic-input.cshtml for the structure of the control and productCtrl.js to see how the input is being configured
     3. Populate the category control using the input.category object and a http call to “api/category”
     4. On Category change, make a http call to populate the Sub Category list. This should call the “GetSubCategories” method from the “Category” api controller, passing the selected category Id. The result should be used to populate the input.subcategory object
     5. On Sub Category change, fetch the Products and pass to the product-list directive. Populate the main tabular grid using Angular bindings. This should filter the results based on the Sub Category chosen, by calling the action “GetProductListBySubCategory” on the “ProductController”
     6. Use the input.minstock object and relevant Angular bindings to change the background of a grid item to red if that Product has a stocklevel lower than the selected minstock value. If it is above the minstock value, the background-colour should be removed
  2. Product/Details.cshtml
     1. Create a Details partial view, Angular route and controller for the Product, displaying all properties for the product
  3. Product/Edit.cshtml
     1. Create an Edit partial view, Angular route and controller for the Product
     2. Implement Update functionality using http post, for this Product
     3. ProductKey should not be editable or visible but should be sent in the update request

**For a Razor/JQuery implementation only:**

1. Product/Index.cshtml
2. This should list Products (use a Partial View for the list), and include Edit and Details links for each row (these pages will be implemented later).
3. The category dropdown ("#category") should be filled by an asynchronous call to “api/category”. The dropdown item value should be “Category.Id”, and the Text should be “Category.Name”.
4. Create a Category dropdown change event. This should call the “GetSubCategories” method from the “Category” api controller, passing the selected category Id. This will fill the Sub Category dropdown (“#subcategory”) with the result. Again, the dropdown item value should be “Category.Id”, and the Text should be “Category.Name”.
5. Create a Sub Category dropdown change event that filters the list of Products based on the selected sub category id by calling the action “GetProductListBySubCategory” on the “ProductController”. This should return a partial view of the list of Products (you will need to replace the existing table)
6. Create a minstock input ("#minstock") change event that loops through each row of ("#producttablebody”) and changes the background colour to red if that Product has a stocklevel lower than the selected minstock value. If it is above the minstock value, the background-colour should be removed.
7. Product/Details.cshtml
8. Create a Details page for the Product, displaying all properties for the product.
9. Product/Edit.cshtml
10. Create an Edit page for the Product.
11. Implement Update functionality for this View.
12. ProductKey should not be editable or visible but should be sent in the update request.