After creating an order, you may want to:

- Display information about the order's status and details to the customers
- Set the order's properties such as the order status or payment results
- Display a "My orders" section to the customers

You can use the **Kentico.Ecommerce** integration package and its built-in methods to cover these scenarios.

Displaying and updating a specific order

You can get any order in the system by its ID. Use the **KenticoOrderRepository** class and its **GetById** method. The method returns an **Order** object that contains the order's properties.

We recommend using a dependency injection container to initialize service instances. When configuring the lifetime scope for the *ShoppingService* and *KenticoOrderRepository* classes, create a separate instance for each request.

```
        public OrderController()
        {
            shoppingService = new ShoppingService();
            orderRepository = new KenticoOrderRepository();
        }
```

Then, you can load whichever order by its ID and configure the order:

Getting an order

```
            // Gets the order based on the order ID
            Order order = orderRepository.GetById(orderID);
```

Setting an order as paid

```
            // Sets the order as paid
            order.SetAsPaid();
```

See the source code of the integration package to get to know all of the available methods and properties.

Displaying a list of orders

Displaying a list of orders is suitable especially for "My orders" sections in customers' profiles. You can get a complete collection of all customer's orders with the **GetByCustomerId** method in the **KenticoOrderRepository** class.

Initialize the **ShoppingService** and **KenticoOrderRepository** classes.

We recommend using a dependency injection container to initialize service instances. When configuring the lifetime scope for the *ShoppingService* and *KenticoOrderRepository* classes, create a separate instance for each request.

```
        public OrderController()
        {
            shoppingService = new ShoppingService();
            orderRepository = new KenticoOrderRepository();
        }
```

Then, get the current customer and list all their orders.

```
            // Gets the current customer
            Customer currentCustomer = shoppingService.GetCurrentCustomer();

            // If the customer does not exist, returns error 404
            if (currentCustomer == null)
            {
                return HttpNotFound();
            }

            // Creates a view model representing a collection of the customer's orders
            OrdersViewModel model = new OrdersViewModel()
            {
                Orders = orderRepository.GetByCustomerId(currentCustomer.ID)
            };
```

In the example, the view model then contains a collection (*IEnumerable<Order>*) of the customer's orders.

Reordering an existing order

If you want to enable the customer to reorder the same order, you can load the order's items and add them to the current shopping cart.

```
            // Gets the order based on its ID
            Order order = orderRepository.GetById(orderId);

            // Gets the current visitor's shopping cart
            ShoppingCart cart = shoppingService.GetCurrentShoppingCart();

            // Loops through the items in the order and adds them to the shopping cart
            foreach (OrderItem item in order.OrderItems)
            {
                cart.AddItem(item.SKUID, item.Units);
            }

            // Evaluates the shopping cart
            cart.Evaluate();

            // Saves the shopping cart
            cart.Save();
```