When you put together the checkout process steps in MVC, one of the usual steps of the checkout process is a step where visitors/customers type their personal or business information. To create the checkout process steps, use the **Kentico.Ecommerce** integration package.

> **Tip**: You can view a sample customer details checkout process step on the sample MVC *Dancing Goat* site.

The following process creates controller actions, models and views that enable customers to:

- Enter their personal and business information.
- Enter their billing address.

  > Similarly, you can also add a different shipping address.

- Select a shipping option they want to use for the order content.

To implement this type of customer details checkout process step:

1. Edit the controller for the checkout process in your MVC project in Visual Studio.
   - All the checkout process steps can share the same controller. See more in Using a shopping cart on MVC sites.

     > The following example uses the **ShoppingService**, **ICustomerAddressRepository** and **IShippingOptionRepository** classes initialized in the controller's constructor. See Using a shopping cart on MVC sites.

2. Create a model for customer details. The model can use the **Customer** model class from the integration package.

```
    public class CustomerModel
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
        public string PhoneNumber { get; set; }
        public string Company { get; set; }
        public string OrganizationID { get; set; }
        public string TaxRegistrationID { get; set; }
        public bool IsCompanyAccount { get; set; }

        /// <summary>
        /// Creates a customer model.
        /// </summary>
        /// <param name="customer">Customer details.</param>
        public CustomerModel(Customer customer)
        {
            if (customer == null)
            {
                return;
            }

            FirstName = customer.FirstName;
            LastName = customer.LastName;
            Email = customer.Email;
            PhoneNumber = customer.PhoneNumber;
            Company = customer.Company;
            OrganizationID = customer.OrganizationID;
            TaxRegistrationID = customer.TaxRegistrationID;
            IsCompanyAccount = customer.IsCompanyAccount;
        }

        /// <summary>
        /// Creates an empty customer model.
        /// </summary>
        public CustomerModel()
        {
        }

        /// <summary>
        /// Applies the model to a customer wrapper.
        /// </summary>
        /// <param name="customer">Customer details to which the model is applied.
</param>
        public void ApplyToCustomer(Customer customer)
        {
            customer.FirstName = FirstName;
            customer.LastName = LastName;
            customer.Email = Email;
            customer.PhoneNumber = PhoneNumber;
            customer.Company = Company;
            customer.OrganizationID = OrganizationID;
            customer.TaxRegistrationID = TaxRegistrationID;
        }
    }
```

3. Create a model for billing addresses. The model can use the **Address** model class from the integration package.

```
    public class BillingAddressModel
    {
        public string Line1 { get; set; }
        public string Line2 { get; set; }
        public string City { get; set; }
        public string PostalCode { get; set; }
        public int CountryID { get; set; }
        public int StateID { get; set; }
        public int AddressID { get; set; }
        public SelectList Countries { get; set; }
        public SelectList Addresses { get; set; }

        /// <summary>
        /// Creates a billing address model.
        /// </summary>
        /// <param name="address">Billing address.</param>
        /// <param name="countryList">List of countries.</param>
        public BillingAddressModel(CustomerAddress address, SelectList countries,
SelectList addresses)
        {
            if (address != null)
            {
                Line1 = address.Line1;
                Line2 = address.Line2;
                City = address.City;
                PostalCode = address.PostalCode;
                CountryID = address.CountryID;
                StateID = address.StateID;
                AddressID = address.ID;
            }

            Countries = countries;
            Addresses = addresses;
        }

        /// <summary>
        /// Creates an empty billing address model.
        /// </summary>
        public BillingAddressModel()
        {
        }

        /// <summary>
        /// Applies the model to an address wrapper.
        /// </summary>
        /// <param name="address">Billing address to which the model is applied.<
/param>
        public void ApplyTo(CustomerAddress address)
        {
            address.Line1 = Line1;
            address.Line2 = Line2;
            address.City = City;
            address.PostalCode = PostalCode;
            address.CountryID = CountryID;
            address.StateID = StateID;
        }
    }
```

4. (Optional) If you want to let your customers have different shipping address from their billing address, create a model for shipping addresses. The model can be the same as the billing address model.
5. Create a model for shipping options. The model can use the **ShippingOptionInfo** class from the **CMS.Ecommerce** namespace.

```
    public class ShippingOptionModel
    {
        public int ShippingOptionID { get; set; }
        public SelectList ShippingOptions { get; set; }

        /// <summary>
        /// Creates a shipping option model.
        /// </summary>
        /// <param name="shippingOption">Shipping option.</param>
        /// <param name="shippingOptions">List of shipping options.</param>
        public ShippingOptionModel(ShippingOptionInfo shippingOption, SelectList
shippingOptions)
        {
            ShippingOptions = shippingOptions;

            if (shippingOption != null)
            {
                ShippingOptionID = shippingOption.ShippingOptionID;
            }
        }

        /// <summary>
        /// Creates an empty shipping option model.
        /// </summary>
        public ShippingOptionModel()
        {
        }
    }
```

6. Add a model that wraps the just created models for the customer and the billing address.

   If you provide a possibility of a different shipping address, add also the shipping address's model.

   You can also add a property for keeping information whether the customer wants the shipping address different from the billing address. For example:

   ```
           public bool ShippingAddressIsDifferent { get; set; }
   ```

7. Add a method to the controller that displays the customer details step. This method requires the **CMS.Globalization** namespace.

```
        /// <summary>
        /// Displays the customer detail checkout process step without any
additional functionality for registered customers.
        /// </summary>
        public ActionResult DeliveryDetails()
        {
            // Gets the current user's shopping cart
            ShoppingCart cart = shoppingService.GetCurrentShoppingCart();

            // If the shopping cart is empty, displays the shopping cart
            if (cart.IsEmpty)
            {
                return RedirectToAction("ShoppingCart");
            }

            // Gets all countries for the country selector
            SelectList countries = new SelectList(CountryInfoProvider.
GetCountries(), "CountryID", "CountryDisplayName");

            // Gets all enabled shipping options for the shipping option selector
            SelectList shippingOptions = new SelectList(shippingOptionRepository.
GetAllEnabled(), "ShippingOptionID", "ShippingOptionDisplayName");

            // Loads the customer details
            DeliveryDetailsViewModel model = new DeliveryDetailsViewModel
            {
                Customer = new CustomerModel(cart.Customer),
                BillingAddress = new BillingAddressModel(cart.BillingAddress,
countries, null),
                ShippingOption = new ShippingOptionModel(cart.ShippingOption,
shippingOptions)
            };

            // Displays the customer details step
            return View(model);
        }
```

If you provide a possibility of a different shipping address, load also the shipping address.

8. Add a method to the controller that processes a filled customer details form. If the processing is successful, the method moves the visitor to the preview step of the checkout process.

```
        /// <summary>
        /// Validates the entered customer details and proceeds to the next
checkout process step with the preview of the order.
        /// </summary>
        /// <param name="model">View model with the customer details.</param>
        [HttpPost]
        public ActionResult DeliveryDetails(DeliveryDetailsViewModel model)
        {
            // Gets the current user's shopping cart
            ShoppingCart cart = shoppingService.GetCurrentShoppingCart();

            // Gets all enabled shipping options for the shipping option selector
            SelectList shippingOptions = new SelectList(shippingOptionRepository.
GetAllEnabled(), "ShippingOptionID", "ShippingOptionDisplayName");

            // If the ModelState is not valid, assembles the country list and the
shipping option list and displays the step again
            if (!ModelState.IsValid)
            {
                SelectList countries = new SelectList(CountryInfoProvider.
GetCountries(), "CountryID", "CountryDisplayName");
                model.BillingAddress.Countries = countries;
                model.ShippingOption.ShippingOptions = new ShippingOptionModel
(cart.ShippingOption, shippingOptions).ShippingOptions;
                return View(model);
            }

            // Gets the shopping cart's customer and applies the customer details
from the checkout process step
            if (cart.Customer == null)
            {
                cart.Customer = new Customer();
            }
            model.Customer.ApplyToCustomer(cart.Customer);

            // Gets the shopping cart's billing address and applies the billing
address from the checkout process step
            cart.BillingAddress = addressRepository.GetById(model.BillingAddress.
AddressID) ?? new CustomerAddress();
            model.BillingAddress.ApplyTo(cart.BillingAddress);

            // Sets the address personal name
            cart.BillingAddress.PersonalName = $"{cart.Customer.FirstName} {cart.
Customer.LastName}";

            // Sets the selected shipping option
            cart.ShippingOption = shippingOptionRepository.GetById(model.
ShippingOption.ShippingOptionID);

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

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

            // Redirects to the next step of the checkout process
            return RedirectToAction("PreviewAndPay");
        }
```

If you provide a possibility of a different shipping address, apply also the shipping address from the shopping cart. For example:

```
            if (model.ShippingAddressIsDifferent)
            {
                // ...
            }
```

9. Add a method to the controller that processes loading of states when a visitor changes the selected country.

```
        /// <summary>
        /// Loads states of the specified country.
        /// </summary>
        /// <param name="countryId">ID of the selected country.</param>
        /// <returns>Serialized display names of the loaded states.</returns>
        [HttpPost]
        public JsonResult CountryStates(int countryId)
        {
            // Gets the display names of the country's states
            var responseModel = StateInfoProvider.GetStates().WhereEquals
("CountryID", countryId)
                .Select(s => new
                {
                    id = s.StateID,
                    name = HTMLHelper.HTMLEncode(s.StateDisplayName)
                });

            // Returns serialized display names of the states
            return Json(responseModel);
        }
```

10. Create a view for the custom details step with a link to a JavaScript file that handles loading of states.

```
<h2>Customer details step</h2>
@using (Html.BeginForm(FormMethod.Post))
{
    <div id="customerDetails">
        <h3>Customer details</h3>
        @Html.EditorFor(m => m.Customer)
    </div>

    <div id="billingAddress">
        <h3>Billing address</h3>
        <div>
            @Html.LabelFor(m => m.BillingAddress.Line1)
            @Html.EditorFor(m => m.BillingAddress.Line1)
        </div>

        <div>
            @Html.LabelFor(m => m.BillingAddress.Line2)
            @Html.EditorFor(m => m.BillingAddress.Line2)
        </div>

        <div>
            @Html.LabelFor(m => m.BillingAddress.City)
            @Html.EditorFor(m =>m.BillingAddress.City)
        </div>

        <div>
            @Html.LabelFor(m => m.BillingAddress.PostalCode)
            @Html.EditorFor(m => m.BillingAddress.PostalCode)
        </div>

        <div class="js-country-state-selector" data-statelistaction='@Url.Action
("CountryStates", "Checkout")' data-countryselectedid='@Model.BillingAddress.
CountryID' data-stateselectedid='@Model.BillingAddress.StateID' data-
countryfield='CountryID' data-statefield='StateID'>
            @Html.LabelFor(m => m.BillingAddress.CountryID)
            @Html.DropDownListFor(m => m.BillingAddress.CountryID, Model.
BillingAddress.Countries, new { @class = "js-country-selector" })
            <div class="js-state-selector-container">
                @Html.LabelFor(m => m.BillingAddress.StateID)
                @Html.DropDownListFor(m => m.BillingAddress.StateID, Enumerable.
Empty<SelectListItem>(), new { @class = "js-state-selector" })
            </div>
        </div>
    </div>

    <div id="shippingOption">
        <h3>Shipping option</h3>
        @Html.LabelFor(m => m.ShippingOption.ShippingOptionID)
        @Html.DropDownListFor(m => m.ShippingOption.ShippingOptionID, Model.
ShippingOption.ShippingOptions)
    </div>

    <input type="submit" value="Continue" />
}
@Scripts.Render("~/Scripts/jquery-2.1.4.min.js")
@Scripts.Render("~/Scripts/countryStateSelector.js")
```

11. Install the **Microsoft.AspNet.Web.Optimization** package using the NuGet Package Manager.

- This integration package ensures that your JavaScript files are rendered correctly.

12. Create a JavaScript file handling the state selector for the selected country. More specifically, it displays the state selector if the selected country contains states. Otherwise, it hides the state selector. Also, the JavaScript dynamically loads the states and adds them to the state selector.

```
»(function () {
    'use strict';

    $('.js-country-selector').change(function () {
        var $countrySelector = $(this),
            $countryStateSelector = $countrySelector.parent('.js-country-state-
selector'),
            $stateSelector = $countryStateSelector.find('.js-state-selector'),
            $stateSelectorContainer = $countryStateSelector.find('.js-state-
selector-container'),
            selectedStateId = $countryStateSelector.data('stateselectedid'),
            url = $countryStateSelector.data('statelistaction'),
            postData = {
                countryId: $countrySelector.val()
            };

        $stateSelectorContainer.hide();

        if (!postData.countryId) {
            return;
        }

        $.post(url, postData, function (data) {
            $countryStateSelector.data('stateselectedid', 0);
            $stateSelector.val(null);

            if (!data.length) {
                return;
            }

            fillStateSelector($stateSelector, data);
            $stateSelectorContainer.show();

            if (selectedStateId > 0) {
                $stateSelector.val(selectedStateId);
            }
        });
    });

    $('.js-country-state-selector').each(function () {
        var $selector = $(this),
            $countrySelector = $selector.find('.js-country-selector'),
            countryId = $selector.data('countryselectedid');

        if (countryId > 0) {
            $countrySelector.val(countryId);
        }

        $countrySelector.change();
        $selector.data('countryselectedid', 0);
    });

    function fillStateSelector($stateSelector, data) {
        var items = '';
```

```
        $.each(data, function (i, state) {
            items += '<option value="' + state.id + '">' + state.name + '<
/option>';
        });

        $stateSelector.html(items);
    }
}());
```

When a visitor gets to the customer details step during their checkout process, they fill in their information (their name and address), and they choose the shipping option they prefer. Finally, they can continue to the next step (usually the preview step).