Table of Contents

CloudCraze 4.8 Training April 2018

Chapter 1. About The Technical Exercises

- The purpose of this documentation is to provide an overview of the extension capabilities and the technical details needed to implement them.
- CloudCraze was designed in a way that allows easy extensibility of its default features.
- The exercises presented are a reflection of the case study: DefaultStore, a hypothetical store.
- Throughout this training, the participants will experience working on both the Front-End and Back-End code components.

The code artifacts used in the exercises are also included in the Training Orgs.

1.1. Audience

- The recommended audience for the training is:
  - Architects (Salesforce, Apex, Integrations, Front-end development)
  - Developers (Front-end development, Salesforce, Apex)
  - Power Users (Salesforce, Apex, Integration)
- Basic understanding of the technology stack (or similar) is enough

1.2. Review of Key Terms

Managed Package

Force.com code consisting of Visualforce pages, components, apex classes, triggers, custom objects, email templates, static resources, etc. that combined makes the CloudCraze core code. This cannot be modified directly and works like a "black box"

Namespace

A naming convention used to help uniquely identify something within an application (e.g. **"ccrz"** is the CloudCraze namespace used in the package)

Subscriber

Custom code that is not part of any managed package (e.g. your own Apex classes, VisualForce pages etc.)

Service Override

Subscriber code that is used to modify (or replace) the SOQL and DML statements run within the CloudCraze Managed Package.

Extension

Subscriber code that is used to extend the functionality of the CloudCraze Managed Package. E.g. `Global class MyCustom_hk_Invoice extends ccrz.cc_hk_Invoice ...`

Configuration Settings

These are used within CloudCraze to control the behavior of CloudCraze on a per-StoreFront/per-Page

basis.

Page Key

Group of settings that are mapped to a specific page or pages (e.g. "HP" for "HomePage", "PLP" for "Product List Page", "PDP" for "Product Detail Page", etc.)

- Managed Package assets can **not** be changed on the org on which they are installed.
- Debug logs for the Managed Package are also hidden
- Unit Tests are not executed for Managed Packages when deploying to production.

F.A.Q.

What if I need an update or a fix for the CloudCraze Managed Package ?

CloudCraze announces patches and updates to its platform periodically (we have major releases almost every quarter!). New versions of the managed package replace and add to the assets already installed. In addition, we also provide Concierge Services and Service Desk as part of our offering, that can be used for support.

What about the CloudCraze Managed Package logs?

They are accessible via the Salesforce License Manager but only to Cloudcraze after being authorized by your admin. You will have to coordinate via our Service Desk to get access to them. Alternatively, there is CCLog.

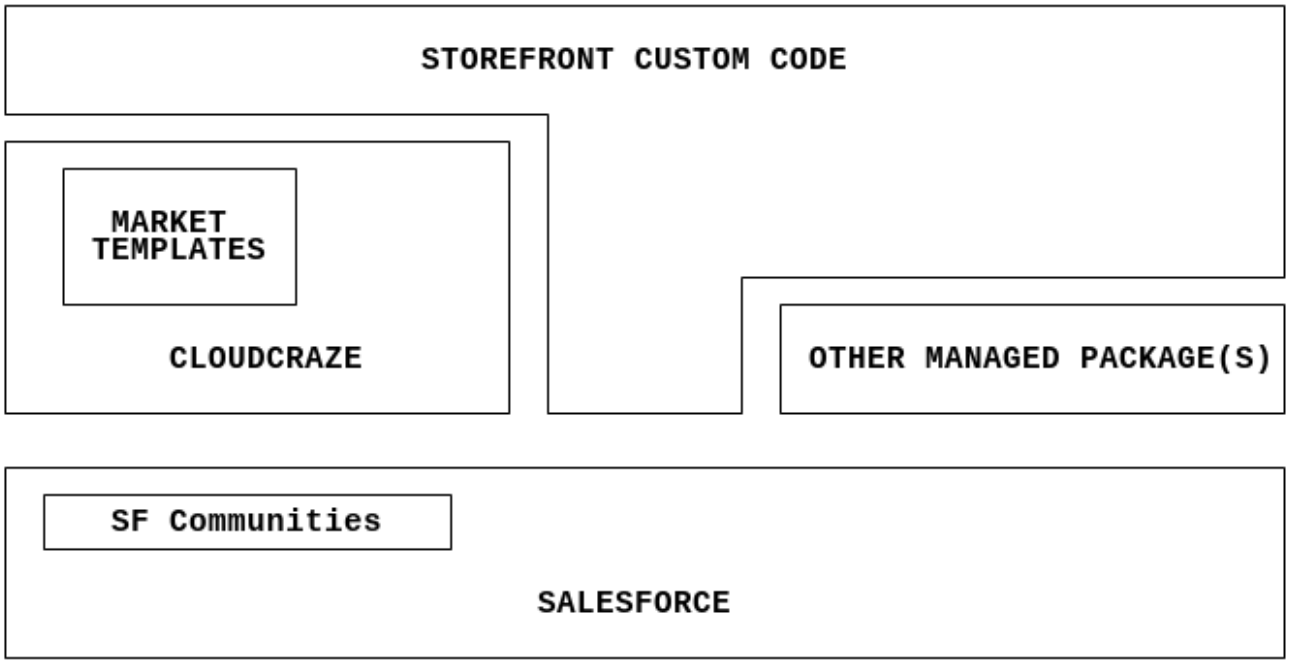
Where can I get the list of Page Configuration Settings and Page Keys?

Our wiki here contains the latest version of our configuration settings.

1.3. Anatomy of a CloudCraze E-Commerce Storefront

CloudCraze is ready to work out of the box, but in more cases than not, extension will be required. Being a managed package, modifications to the behavior of the application occur through **extensions** and **overrides** via custom code.

In the sample diagram below, DefaultStore storefront extended CloudCraze and its Market Templates' functionality, as well as direct SalesForce customizations (e.g. additional SF objects and Scheduled Jobs) and other managed packages.



A Typical CloudCraze E-Commerce implementation will be built on top of one or more SalesForce communities, and may include additional functionality via Market Templates, custom code, and 3rd party integrations. Other than Salesforce itself, CloudCraze does **not** depend on external databases or servers.

1.4. Technology Stack

In order to provide a rich e-commerce experience that is also extensible and maintainable, CloudCraze chose the following stack:

Front End: changes to page layouts, components, flows, etc.

- Bootstrap 3: responsive design CSS Framework.
- Backbone.JS and Underscore.js: Model-View Javascript library and utility.
- Handlebars: semantic HTML template engine.
- jQuery: popular JavaScript library.
- Other libraries and utilities included:
  - Zebra Date Picker
  - Font-Awesome
  - NoUISlider
  - jQueryUI
  - accounting
  - Backbone paginator

It is crucial you spend time studying these diagrams in the appendix to best understand how the CloudCraze front-end is built and how best you can extend or customize features to handle your targeted use cases. Click here to view the diagrams.

Back End: changes to the logic for products, orders, payments, carts, categories, etc.

- SalesForce: SalesForce is the Web Server, Database, Identity Manager, Integration Server, etc.

F.A.Q.

Why Backbone?

We wanted to have a good and mature js framework that did not tie the front-end to any particular design paradigm (e.g. SPAs, build servers) so it will be less difficult to extend or replace if needed. Backbone provided the flexibility to do so.

Chapter 2. User Interface

- The first part of this course will cover the User Interface.
- Requisites are understanding HTML, CSS, JavaScript as well as basic UI/UX concepts such as themes and responsive design.

2.1. Review of Key Terms

CloudCraze Theme

A static resource file that consists of a collection of css, images, Javascript and other resources that defines the look and feel of a CloudCraze storefront. It is primarily used to include additional styling that complements the default theme in CloudCraze.

CloudCraze Template (Layout)

Base layout that defines overall structure of the page (e.g. column formatting and component placement). Some of the default CloudCraze templates are:

1. cc_tmpl_OneColRD
2. cc_tmpl_TwoColRD
3. cc_tmpl_ThreeColRD

CloudCraze Page (CCPage)

A Visual Force Page included with Cloudcraze, extendable via subscriber code.

Page Include

An "add-on" that can be placed in your storefront page or pages to extend or replace existing functionality.

Subscriber Page

A fully customizable VisualForce page, called from the "CCPage" page.

Subscriber Template

Custom layouts for the pages that are built by extending the default `cc_tmpl_SubscriberTemplate` template.

`ccrz.cc_hk_UserInterface` extensions need to be used to modify or inject additional content into the `<head>`.

Even though two slots exist (Header Include Begin and Header Include End), Page Includes should not be used to modify or inject additional content into the HTML `<head>`.

Page Include slots leverage the `<apex:include>` mechanism in VisualForce which wraps the Subscriber VF Page in `<span>` tags. Most modern browsers will not support this and will instead terminate the `</head>` in the DOM at the first sight of a `<span>`, effectively pushing any content from the Page Include into the `<body>` of the DOM.

- The list of CloudCraze ready to use Subscriber Templates can be found here
- When using custom Subscriber Templates, remember to setup two configuration settings: the first one to activate cc_tmpl_SubscriberTemplate as your page layout and another one for your custom template itself.
- For Subscriber Pages, a page key needs to be registered in advance in order for it to be called from within CloudCraze. Check out UI Subscriber Pages for more info.
- There is also a base template called Storefront Template (cc_tmpl_storefront). This one is read-only and contains common elements for all pages (header, footer, page include place holders, etc.). It can't be modified directly.

2.2. UI Recommendations

In order to provide maximum upgrade coverage and support, CloudCraze recommends trying to make your customizations via **Configuration** first before considering **extensions via subscriber code**.

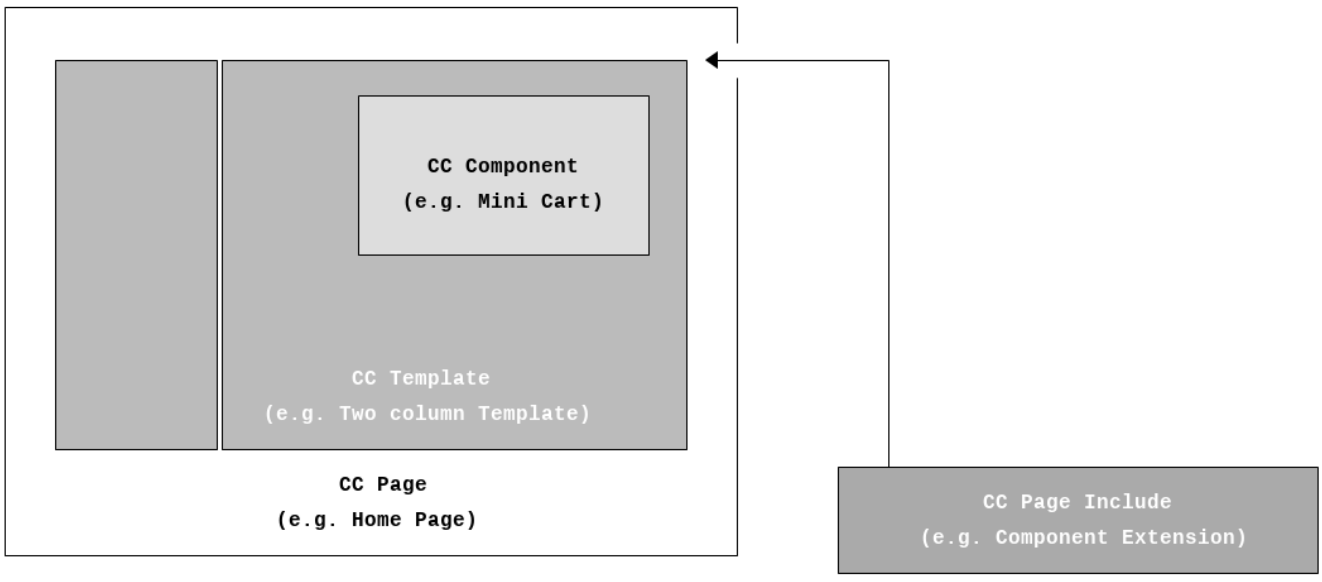
Configuration

This includes branding or skinning made using configuration settings and changes to the Theme files (CSS, js, images, etc.)

Extensions via Subscriber Code

Branding or skinning in addition to customization. This could be changes in the form of udpates to Templates, CC Pages, components, page includes etc.

2.3. UI Overview Diagram



UI Overview:

- Every **CloudCraze Page** (e.g. Home Page) is linked to a **CC Template** (e.g. Two-column template) which

defines the layout for its contents.

- CloudCraze pages also includes one or more **VisualForce Components** (e.g. Mini-Cart) which are modular and can be shared across multiple pages and storefronts as needed.
- In addition, CloudCraze provides **Page Includes** (e.g. a Component Extension) which can be used to extend or replace existing functionality within a page or pages in a StoreFront.
- They share a common **Theme** (e.g. DefaultStore Theme) that defines the overall look and feel of the Storefront pages.

2.4. UI Theming (CloudCraze Themes)

- CloudCraze themes are enabled at the Storefront level.
- A CloudCraze theme is primarily made up of CSS rules, JavaScript and images.
- Additional folders can be included and referenced as needed (e.g. additional fonts, localization folders, etc.).

Folder Structure of a CloudCraze Theme

A typical CloudCraze theme consists of the following folder Structure:

- CC_MyCompany_Theme
  - css (for Bootstrap 2)
    - styles.css: custom styles for Bootstrap 2 goes here
    - bootstrap.css: bootstrap2 css
    - bootstrap_overwrite.css: bootstrap overrides
  - css3 (for Bootstrap 3)
    - styles.css: custom styles for Bootstrap 3 goes here
  - images
  - js
    - uiproperties.js: useful to override uiproperties globally (e.g. Handlebars templates referenced in backbone views)

- Cloudcraze Themes are enabled via the CC Admin link in the Salesforce org.
- These themes are SalesForce static resources, and they need to have **"Theme"** as part of its name to be picked up by CCAdmin (e.g. `CC_MyCompany_Theme` or `MyCompany_Theme_ResponsiveDesign`)
- Static resources allow you to upload content that you can reference in your pages, including archives (such as .zip and .jar files), images, style sheets, JavaScript, and other files.

- When creating your own theme, sometimes it is easier to use an existing theme as a starting point (e.g. `CC_Theme_CloudburstRD`) and then customize the theme based on your business needs.

F.A.Q.

What if I want to include libraries from a CDN or additional files?

CloudCraze provides several mechanisms to do so, a few of which are:

1. Extending the Header or Footer components
2. cc_hk_userinterface
3. Page Includes
4. Subscriber Pages

2.4.1. Exercise: Working with CloudCraze Themes

In this exercise, you'll learn how to upload a custom theme and make modifications to it.

Use Case

1. Upload a custom theme for the DefaultStore storefront
2. Make the theme compatible with Bootstrap 3
3. Modify the navbar background color across all your pages

You can skip ahead to the next use case if you've already enabled and tested your theme in a previous chapter. Also, while you don't need deep SalesForce or CSS knowledge, ideally you should be familiar with the concept of uploading/downloading Static Resource files.

2.4.1.1. Uploading and activating a custom theme.

1. Download **ccrz__CC_Theme_CloudBurstRD.zip** from the training wiki space
2. Upload ccrz__CC_Theme_CloudBurstRD in your Salesforce Org:
   a. In your Salesforce Org, click on **Setup** (the link is near the top right corner).
   b. On the left pane, click on **Build → Develop → Static Resources**
   c. Click the **New** button.
   d. Enter the following on the static resources screen:

      Name

      ccrz__CC_Theme_CloudBurstRD

Description

DefaultStore Theme

File

click on the "Browse" button and locate and select the **ccrz__CC_Theme_CloudBurstRD.zip** file downloaded in step 1.

Cache Control

Public

e. Finally, click the **Save** button to submit your changes.

3. Enable the ccrz__CC_Theme_CloudBurstRD theme:
   a. In your Salesforce Org, click **CC Admin** from the main tab.
   b. On the right side dropdown select the **DefaultStore** Storefront.
   c. On the left side bar, near the bottom, under the "Appearance" section, click on **Themes**.
   d. From the list of themes, locate "ccrz__CC_Theme_CloudBurstRD" and click on **Enable**.
4. Create and Activate a new Configuration Cache:
   a. On the right side dropdown, click on the **Global Settings** link.
   b. On the left side bar, select **Configuration Cache Management**.
   c. Click on **Build New**. After a few seconds, click on **Refresh List**. A new configuration cache should have been built. Click on **"Activate"** to enable it.
5. Test your theme:
   a. In your SalesForce Org, click on **Contacts** from the main tab.
   b. Search for **Jon Amos**. Click on the name to open the contact info.
   c. In the Contact Detail Section, select **Manage External User**, and from the dropdown, select **Log in to Community as User**.
6. Your new theme should take effect.

F.A.Q.

What is the configuration cache?

The configuration cache is a collection of indexed configuration settings (plus the CC theme settings) that is used by CloudCraze to keep the thousands of storefront settings rapidly accessible across the sites.

- When creating a new configuration cache, it is good practice to deactivate all the older ones.
- Also, it is recommended to only keep a limited number of configuration caches and remove the rest to prevent hitting any SalesForce limits.

Keep in mind that while most configuration settings can be set at the storefront and page level, the **configuration cache is global**.

2.4.1.2. Enabling the Bootstrap 3 theme (Cloudcraze UI v003)

1. Switching the User Interface Extension settings:
    a. In your Salesforce Org, click **CC Admin** from the main tab.
    b. On the right side dropdown select the **DefaultStore** Storefront.
    c. On the left side bar, near the bottom, under the "Appearance" section, click on **Themes**.
    d. Under "User Interface Extension Settings for DefaultStore" replace the 'User Interface Extension API Class' from `cc_hk_UserInterface:v002` to `cc_hk_UserInterface:v003`.
    e. Save your changes
2. Update the Configuration settings for Bootstrap 3:
    a. In your Salesforce Org, click **CC Admin** from the main tab bar.
    b. On the right side dropdown select the **DefaultStore** Storefront.
    c. On the left side bar, under Settings, select **Configuration Settings**.
    d. In the Module dropdown, select **User Interface**. One of the settings displayed should be "Template Version", with value of "classic". Select the **Override** link for that row.
    e. In the modal window, replace the attribute value with `boot3` as shown below

    Module
    User Interface

    Configuration
    Template Version

    Page
    All

    Attribute Value
    boot3

    f. Click on **Create**
3. Create and Activate a new Configuration Cache:
    a. On the right side dropdown, click on the **Global Settings** link.
    b. On the left side bar, select **Configuration Cache Management**.
    c. Click on **Build New**. After a few seconds, click on **Refresh List**. A new configuration cache should have been built. Click on **"Activate"** to enable it.
4. Test your theme:
    a. In your SalesForce Org, click on **Contacts** from the main tab bar.

b. Search for **Jon Amos**. Click on the name to open the contact info.

c. On the Contact Detail Section, click on **Manage External User**, and from the dropdown, select "Log in to Community as User".

5. Your new theme should take effect. Notice the new look and feel.

CloudCraze UI v003 is compatible with Bootstrap 3 themes. This makes it a trivial exercise to apply a theme downloaded from popular theme repositories like Boostrap Official Themes or Bootswatch.

2.4.1.3. Modifying the custom theme

1. Unzip the **ccrz__CC_Theme_CloudBurstRD.zip** file that is currently enabled in your storefront.
2. Open **css3/styles.css** in your favorite editor.
3. Modify the background and border colors:
   a. In the styles.css file, enter the following CSS rule:

```
.navbar-inverse {
  background-color: #004488;
  border-color: #004488;
}
```

4. Save the file.
5. Zip your theme resource folder as a new file (e.g. **CC_Theme_DefaultStore_New.zip**).

Be sure to create your new file by zipping the contents in the parent folder. E.g Select css, css3, en_US et.al and then compress these folders into a zip file.

6. Upload your new Theme:
   a. In your Salesforce Org, click on **Setup** (the link is near the top right corner).
   b. Type **static resources** in the quick find box on the left pane.
   c. Select **Static resources** from the **Build → Develop → Static Resources** menu.
   d. Select **New** and enter the following on the static resources screen:

Name

CC_Theme_DefaultStore_New

Description

DefaultStore Theme

File

click on the "Browse" button, then locate and select the **CC_Theme_DefaultStore_New.zip** file you just created.

Cache Control

Public

e. Select the **Save** button to persist your changes.

7. Enable your new theme:
   a. In your Salesforce Org, select **CC Admin** from the main tab.
   b. On the right side dropdown select the **DefaultStore** Storefront.
   c. On the left side bar, near the bottom, under the "Appearance" section, click on **Themes**.
   d. From the list of themes, locate **CC_Theme_DefaultStore_New** and click on **Enable**.
8. Create and Activate a new Configuration Cache:
   a. On the right side dropdown, click on the **Global Settings** link.
   b. On the left side bar, select **Configuration Cache Management**.
   c. Click on **Build New**. After a few seconds, click on **Refresh List**. A new configuration cache should have been built. Click on **"Activate"** to enable it.
9. Test your theme:
   a. In your SalesForce Org, click on **Contacts** from the main tab.
   b. Search for **Jon Amos**. Click on the name to open the contact info.
   c. On the Contact Detail Section, select **Manage External User**, and from the dropdown, select "Log in to Community as User".
10. Your new theme navbar background should take effect - you'll notice the header and footer section now have a blue background.
11. Feel free to revert your changes or keep the new color scheme.

- Remember that there are two CSS folders in a CloudCraze Theme: css and css3. "css" is for Bootstrap 2 overrides, while "css3" is for Bootstrap 3 overrides. Keep this in mind when writing your themes.
- Since CC Themes are dependent on a certain file and folder structure, make sure that when zipping the Theme resource file you are not creating an extra parent folder within it.

2.5. UI Layouts (Subscriber Templates)

- The CloudCraze managed package by default has several layout templates that can be applied to your page (or pages). These templates accommodate popular design interfaces and a few of them are listed below:
  - One column layout: cc_tmpl_OneColRD
  - Two column layout: cc_tmpl_TwoColRD
  - Three column layout: cc_tmpl_ThreeColRD

- Layouts also include placeholders for CloudCraze widgets. The widgets can be enabled or disabled via configuration settings on their respective pages.
- It is also possible to create custom layouts (subscriber templates)

F.A.Q.

What are common scenarios for creating subscriber templates?

Subscriber templates are useful when you want to modify the overall layout of your storefront across multiple pages. E.g. when you want to add a custom widget (or move an existing one) on numerous pages

2.5.1. Exercise: Subscriber Template

In this exercise, we'll modify an existing page to have a different layout. We'll show you how to enable and disable widgets and how to create a Subscriber Template.

Objectives

1. Configure the Home Page (HP) of DefaultStore to use 3 columns instead of 2.
2. Remove the Product Search widget from the Home Page (HP).
3. Move the My Cart widget and put it on top of the left side Navigation via a custom Subscriber Template

2.5.1.1. Configuring the Home Page (HP) to a Three-column layout

1. Modify the Configuration Setting for the Home Page (HP) Template:
   a. In your Salesforce Org, click **CC Admin** from the main tab.
   b. On the right side dropdown select the **DefaultStore** Storefront.
   c. On the left side bar, near the top, under "Settings" section, click on **Configuration Settings**.
   d. To locate what needs to be changed faster, use the filters at the top:

      Module
      Template

      Page
      Home

   e. Modify the Configuration Setting from a "cc_tmpl_TwoColAltRD" to a **"cc_tmpl_ThreeColRD"** by clicking on the value link.
2. Create and Activate a new Configuration Cache:
   a. On the right side dropdown, click on the **Global Settings** link.

b. On the left side bar, select **Configuration Cache Management**.

c. Click on **Build New**. After a few seconds, click on **Refresh List**. A new configuration cache should have been built. Click on **"Activate"** to enable it.

3. Test your changes:

a. In your SalesForce Org, click on **Contacts** from the main tab bar.

b. Search for **Jon Amos**. Click on the name to open the contact info.

c. On the Contact Detail Section, click on **Manage External User**, and from the dropdown, select "Log in to Community as User".

d. Verify that the home page now uses a 3-column layout as opposed to 2.

2.5.1.2. Removing the Product Search widget from the Home Page (HP)

1. Modify the Configuration Setting for the Home Page (HP) Template:

a. In your Salesforce Org, click **CC Admin** from the main tab.

b. On the right side dropdown select the **DefaultStore** Storefront.

c. On the left side bar, near the top, under "Settings" section, click on **Configuration Settings**.

d. Click on **New** button to create a new Configuration Setting and enter the following:

Module
Search Box

Configuration
Enabled

Page
Home (HP)

Value
FALSE

When filling out the value for the `Page` input field, you should keep the following in mind. Start out by typing the name of the page of interest. E.g Type `Home` and then select the appropriate value from the autocompleted suggestions in the input field.

2. Create and Activate a new Configuration Cache:

a. On the right side dropdown, click on the **Global Settings** link.

b. On the left side bar, select **Configuration Cache Management**.

c. Click on **Build New**. After a few seconds, click on **Refresh List**. A new configuration cache should have been built. Click on **"Activate"** to enable it.

3. Test your changes:

a. In your SalesForce Org, click on **Contacts** from the main tab.

b. Search for **Jon Amos**. Click on the name to open the contact info.

c. In the Contact Detail Section, select **Manage External User**, and from the dropdown, select "Log in to Community as User".

d. Verify that the Product Search widget is no longer visible on the home page.

2.5.1.3. Repositioning the Cart widget and moving it to the top of the left side navigation via a Subscriber Template

1. Take note of the current location of the cart widget
2. Open the SalesForce Developer Console:
   a. In your SalesForce Org, click the **Developer Console** link under the dropdown menu **under your name** (on the top right side of the page).
   b. A new window should pop-up.
   c. Alternatively, you can use this URL: https://<YOUR_SALESFORCE_DOMAIN>/_ui/common/apex/debug/ApexCSIPage
3. Create a New VisualForce page (call it "cc_tmpl_Training"):
   a. On the Developer Console menu, select: **File → New → VisualForce Page**
   b. In the popup window (New Apex Page), enter **"cc_tmpl_Training"** as your page name and click **ok**.
4. Enter the following VisualForce code to define the new template:
   a. Remove the boilerplate code (delete the contents of the page).
   b. Enter the following:

```
<apex:page id="cc_tmpl_Training" docType="html-5.0"
  sidebar="false" showHeader="false"
  standardStylesheets="false" applyHtmlTag="false">   ①
  <apex:define name="htmlbody">   ②
    <div class="container cc_main_container cc_tmpl_Training">   ③
      <div class="row cc_main_row">
        <div class="col-md-3 cc_left_col">
          <apex:insert name="WidgetBoxL" />
          <apex:insert name="MiniCartBox" /> ④
          <div id="categories-left-nav"></div>
          <div class="filterContainer"></div>
          <div class="promotion-box-LeftNav"></div>
          <apex:insert name="LeftNavX" />
        </div>
        <div class="col-md-6 cc_main_content_col">
          <div class="row cc_main_content_row">
            <apex:insert name="WidgetBoxC" />
            <apex:insert name="Banner" />
            <div class="promotion-box-Banner"></div>
            <div class="effwig"></div>
            <apex:insert name="ProductSpotlight" />
            <apex:insert name="body" />
            <apex:insert name="PromosCenter" />
            <apex:insert name="FeaturedProducts" />
            <apex:insert name="GuideProducts" />
            <apex:insert name="UpsellProducts" />
            <apex:insert name="ProductReviews" />
            <apex:insert name="CenterX" />
          </div>
        </div>
        <div class="col-md-3 cc_right_col right_column">
          <div class="effright"></div>
          <div class="search-box-RightNav" />
          <apex:insert name="WidgetBoxR" />
          <div class="widgetSection"></div>
          <apex:insert name="MiniQuickOrderBox" />
          <div class="promotion-box-RightNav"></div>
          <apex:insert name="RightNavX" />
        </div>
      </div>
    </div>
  </apex:define>
</apex:page>
```

① These apex:page attributes are required to prevent SalesForce from applying its default standard HTML styling.

② **htmlbody** is picked by cc_tmpl_SubscriberTemplate to render its contents.

③ We'll define a 3-column layout using standard Bootstrap 3.

④ This is the insert for the Cart widget (minicart).

c. Save the page (On the Developer Console menu, click: **File → Save**)

5. Modify the Configuration Setting for the Home Page (HP) Template to use the custom Subscriber Template:

a. In your Salesforce Org, click **CC Admin** from the main tab bar.

b. On the right side dropdown select the **DefaultStore** Storefront.

c. Select **Configuration Settings** from the left side bar

d. To locate what needs to be changed faster, use the filters at the top:

Module

Template

Page

Home

e. Click on the **Value** link to change it to use the Subscriber Template container. Change it from: **cc_tmpl_ThreeColRD** to: **cc_tmpl_SubscriberTemplate**.

f. Next, click on **New** to assign the newly created template to the Home page. Enter the following information:

Module

Template

Configuration

page

Page

Home

Attribute Value

c__cc_tmpl_Training

If the **page** option isn't available, create a metadata for it as described in the steps below, but ensure you revisit this step afterwards.

g. Create a metatadata entry for **page** like so:

i. On the right side dropdown, click on the **Global Settings** link.

ii. Select **Configuration Modules** from the menu on the left hand side.

iii. Scroll through the list of available options and select **Template**

iv. Select **New** in the configuation metadata section and entter the following:

Name

Page

API Name

page

Decription

Subscriber template Visualforce page.

Revisit the previous step if you created the metadata as described above.

6. Create and Activate a new Configuration Cache:
    a. On the right side dropdown, click on the **Global Settings** link.
    b. On the left side bar, select **Configuration Cache Management**.
    c. Click on **Build New**. After a few seconds, click on **Refresh List**. A new configuration cache should have been built. Click on **"Activate"** to enable it.
7. Test your changes:
    a. In your SalesForce Org, click on **Contacts** from the main tab bar.
    b. Search for **Jon Amos**. Click on the name to open the contact info.
    c. On the Contact Detail Section, click on **Manage External User**, and from the dropdown, select "Log in to Community as User".
    d. Open the home page and confirm that the My Cart widget shows on top of the Left Side Navigation.

To expedite the creation of a subscriber template, you could use the Developer Console to view the contents of any of the default Cloudcraze templates. You can then make your edits by adding or removing sections from it as needed.

When adding subscriber pages, remember that two configuration settings are needed (it is quite easy to forget one of them):

1. The template **Name** setting, with the value of `cc_tmpl_SubscriberTemplate` to indicate CloudCraze to use a custom layout.
2. The template **Page** name setting, set to the name of the Subscriber Page, ALWAYS prefixed with `c__` (with two underscores).

2.6. UI Custom Pages (Subscriber Pages)

Cloudcraze comes with more than a dozen ready-to-use VisualForce pages: Home Page, Products list, Product details, Checkout, Cart, etc. as seen below:

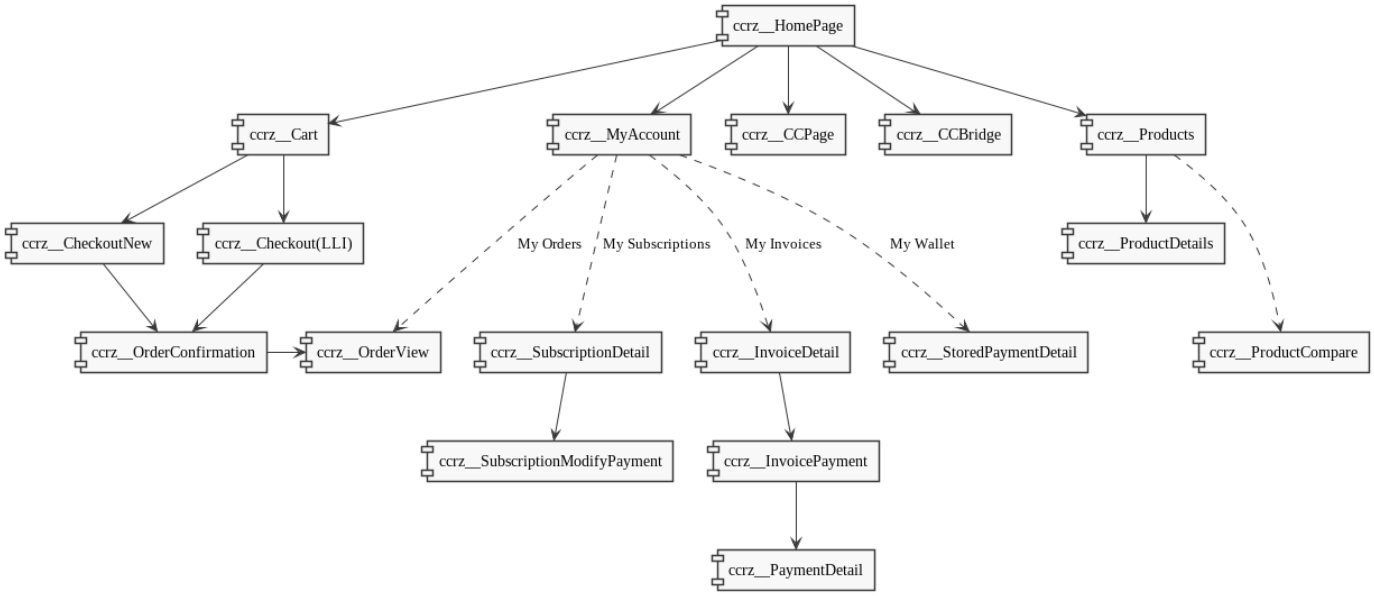


Figure 1. Default CloudCraze Storefront Site Map (excluding administration screens, e.g. Login and Registration pages)

Most of the functionality on these pages can be extended through configuration settings and page includes.

In cases where the default page offering doesn't meet the business needs, Cloudcraze provides a mechanism to create a fully customizable page called Subscriber Page.

A Subscriber page is comprised of 4 main elements:

1. **CC Subscriber Page Name**: This is used to identify the subscriber page in the SalesForce Org.
2. **Page Include**: This refers to the custom VisualForce page that contains the markup for the page. It is always prefixed with `c__` to indicate it is custom.
3. **Page Key**: This is used in conjunction with ccrz__CCPage to render the contents of the Subscriber page in the browser.
4. **Storefront**: This is the name of the storefront where the subscriber page will be active.

In addition, a Subscriber Page usually follows this URL pattern:

```
https://<STOREFRONT_URL>/ccrz__CCPage?pagekey=<Page_Key>
```

F.A.Q.

What are some common scenarios for creating subscriber pages?

Subscriber pages are used for larger bodies of content or functionalities that are not provided by default in CloudCraze: CMS pages (e.g. About Us, Contact Us, etc.). Another possible candidate for this would be E-commerce pages (e.g. Promotions and Deals) and perhaps a UI overhaul (e.g. Single Page apps).

For more info on the subscriber page object, including additional properties that can be added, see Subscriber Pages Wiki

2.6.1. Exercise: Creating a Custom Page

Use Case

1. DefaultStore would like to have an additional page to provide customers with contact information.
2. The page should share the same look and feel as the rest of the storefront site.

2.6.1.1. Creating a Subscriber Page

1. Open the SalesForce Developer Console:
   a. In your SalesForce Org, click the **Developer Console** link under the dropdown menu **under your name** (on the top right side of the page).
   b. A new window should pop-up.
   c. Alternatively, you can use this URL: https://<YOUR_SALESFORCE_DOMAIN>/_ui/common/apex/debug/ApexCSIPage
2. Create a New VisualForce page (call it "ccTraining_SP_ContactUs"):
   a. On the Developer Console menu, select: **File → New → VisualForce Page**
   b. In the popup window (New Apex Page), enter **"ccTraining_SP_ContactUs"** as your page name and click **ok**.
3. Enter the following VisualForce code to define the new template:
   a. Remove the boilerplate code (delete the contents of the page).
   b. Enter the following:

```
<apex:page id="ccTraining_SP_ContactUs" docType="html-5.0"
  sidebar="false" showHeader="false"
  standardStylesheets="false" applyHtmlTag="false">   ①
    <div class="row">  ②
        <div class="col-md-offset-3 col-md-9">
            <h3>Contact Us</h3>
            <div>
                101 N Upper Wacker<br/>
                Chicago, IL 60606 USA<br/>
                (866) 217-3210
            </div>
            <p/>
        </div>
    </div>
</apex:page>
```

① These apex:page attributes are required to prevent SalesForce from applying its default standard HTML styling.

② Custom **Contact Us** details.

This simple example demonstrates adding arbitrary HTML inside of the CloudCraze standard UI components. For more complex components which handle user interaction, it is recommended to keep your implementation consistent with the CloudCraze approach for rendering front-end components client-side.

For more information on how the frontend frameworks interact with one another, refer to these diagrams in the Appendix.

c. Save the page (On the Developer Console menu, click: **File → Save**)

4. Assigning a Subscriber Page to the Visual Force page:
   a. In your Salesforce Org, click on **CC Subscriber Page** from the main tab bar
   b. Click the **New** button.
   c. In the New CC Subscriber Page screen, enter the following:

      CC Subscriber Page Name

      `cc_ContactUs`

      Page Include

      `c__ccTraining_SP_ContactUs`

      Page Key

      `ContactUs`

      Storefront

      `DefaultStore`

   d. Finally, click the Save button to submit your changes.
5. Assigning a Page UIKey reference to the subscriber page:
   a. Go back to the developer console (opened in the first step above)
   b. On the Developer Console menu, click: **Debug → Open Execute Anonymous Window** (alternatively, you could also press **CTRL+E**)
   c. Enter the following Apex code:

```
ccrz.cc_util_Reflection.upsertPageUIKey('cc_ContactUs','ContactUs','Contac
t Us');
```

   d. Click on the `Execute` button to run the code
6. Adding a menu link to the subscriber page.
   a. In your SalesForce Org, click on CC Menu from the main tab bar.
   b. Click on **New** button.
   c. In the New CC Menu screen, enter the following:

DisplayName

`Contact Us`

Store ID

`DefaultStore`

Link Type

`URL`

URL

`/DefaultStore/ccrz__CCPage?pagekey=ContactUs`

d. Next, click the Save button to submit your changes.

e. Now, click on **CC Admin** from the main tab bar.

f. On the left side bar, near the top, under "Global Settings" section, click on **Indexing**.

g. Click on **Refresh Menu Cache** button.

7. Test your new Subscriber Page:

   a. Search for Jon Amos. Click on the name to open the contact info.

   b. On the Contact Detail Section, click on Manage External User, and from the dropdown, select **"Log in to Community as User"**.

   c. On your storefront site, click on **"Contact Us"** under the main menu.

Bonus Points:

1. Improve the look and feel by manipulating the HTML and the Theme's CSS.
2. Make the content configurable, by using either CC Content or CC Page Labels.
3. Add a controller for custom logic.

2.7. UI Page Sections (Page Includes)

Since CloudCraze is a managed package, its Visualforce pages cannot be directly modified in your Salesforce org. To enable the extensibility of the product by subscribers, it's since provided a mechanism that allows one to inject content into the Visualforce pages through special page sections or slots. This mechanism is called **Page Includes**.

Page Includes leverages the `<apex:include>` mechanism in VisualForce which wraps the Subscriber VF Page in `<span>` tags.

2.7.1. Page Include Slots

From top to bottom, the following page include sections or slots are available:

1. HTML Body Include Begin
2. Body Include Begin
3. Body Include End
4. HTML Body Include End

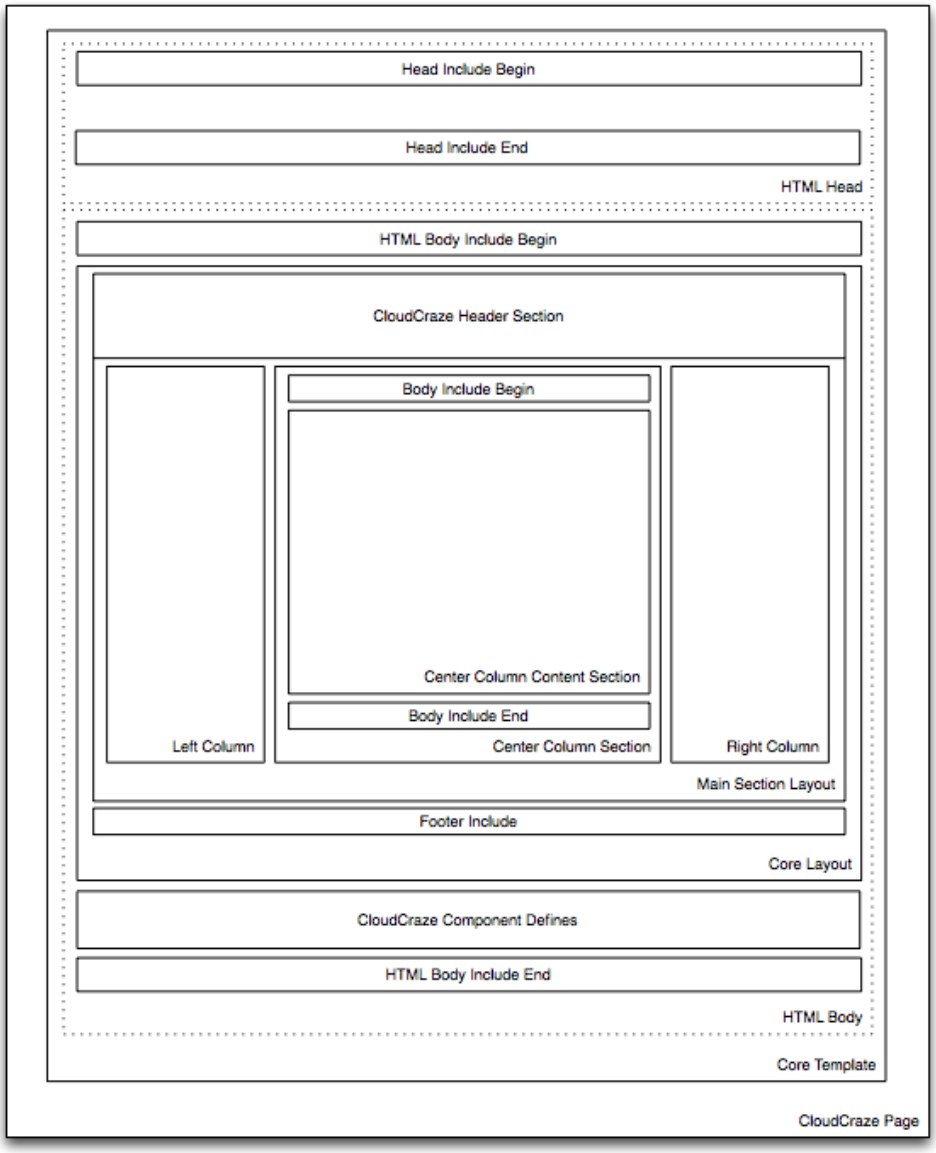


Figure 2. Page Include Slots

- The most commonly used Page Include slots are the `Body Include Begin (BIB)` and `Body Include End (BIE)`

- Page Include values are set via Configuration Settings.
- A Page Include is a VisualForce page, therefore it supports VisualForce functionality, like components and controllers.

- Each Page Include slot can have **up to** one Page Include.

F.A.Q.

What are common scenarios for using page includes?

Page includes are used to extend or override the look and feel of an existing CloudCraze page or to add new functionality.

When is it not recommended to use page includes?

They are not recommended for modifying the `<head>` tag (e.g. to add custom meta tags or load js/css libraries). For those cases, it's better to use the `ccrz.cc_hk_UserInterface` class.

2.7.2. Exercise: Page Include

In this exercise, we'll show you how to create and activate a page include for a storefront.

Use Case

1. DefaultStore would like to have an additional widget on the home page that displays the top 5 news from reddit.
2. The info should be retrieved by invoking their open and free web service here: `https://www.reddit.com/r/news/top/.json?limit=5`

2.7.2.1. Creating a Page Include

1. Open the SalesForce Developer Console:
   a. In your SalesForce Org, click the **Developer Console** link under the dropdown menu **under your name** (on the top right side of the page).
   b. A new window should pop-up.
   c. Alternatively, you can use this URL: https://<YOUR_SALESFORCE_DOMAIN>/_ui/common/apex/debug/ApexCSIPage
2. Create a New VisualForce page (call it "ccTraining_PI_TopNews"):
   a. On the Developer Console menu, select: **File → New → VisualForce Page**
   b. In the popup window (New Apex Page), enter **"ccTraining_PI_TopNews"** as your page name and click **ok**.
3. Enter the following VisualForce code to define the new template:
   a. Remove the boilerplate code (delete the contents of the page).
   b. Enter the following:

```
<apex:page id="ccTraining_PI_TopNews" docType="html-5.0"
  sidebar="false" showHeader="false"
  standardStylesheets="false" applyHtmlTag="false">

  <!-- HTML CONTAINER -->
  <h3>Meanwhile in the News...</h3>
  <ol class="anchorClass"></ol>                                                    ①
  <p>
    <small>News Content provided by
      <a href="https://www.reddit.com" target="_blank">Reddit</a>
    </small>
  </p>

  <!-- HANDLEBARS TEMPLATE -->
  <script type="text/template" id="TopNews_HBTemplate">
    {{#each this.children}}                                                        ②
      <li>
        <a href="{{data.url}}" target="_blank">{{data.title}}</a>
      </li>
    {{/each}}
  </script>

  <!-- JS CODE -->
  <script type="text/javascript">
    jQuery(function($) {
        CCRZ.subsc        = CCRZ.subsc || {};
        CCRZ.subsc.views  = CCRZ.subsc.views || {};
        CCRZ.subsc.views.TopNews_BBView = CCRZ.CloudCrazeView.extend({          ③
          templateName: CCRZ.util.template('TopNews_HBTemplate'),              ④
          renderSelector: ".anchorClass",                                      ⑤
          viewName: "TopNews_BBView",
          managedSubView: true,
          renderDesktop: function(){                                           ⑥
            this.renderView(this.templateName, this.renderSelector);
          },
          renderPhone : function() {
            this.renderView(this.templateName, this.renderSelector);
          },
          renderView: function(templateName,renderSelector) {
            var v = this;
            v.setElement(renderSelector);                                      ⑦
            webServiceURL="https://www.reddit.com/r/news/top/.json?limit=5"; ⑧
            $.ajax({
              url:webServiceURL,
              success:function(data){
                var modelData = data;
                v.$el.html(templateName(modelData.data));                      ⑨
              }
            });
          }
        });
        TopNews = new CCRZ.subsc.views.TopNews_BBView({});                     ⑩
        TopNews.render();
      });
    </script>

</apex:page>
```

① `anchorClass` is the parent element where we will like our rendered handlebar template to appear.

② `#each` is a **Handlebars helper function** which iterates across the list of items named **children**

③ Register a custom backbone view (TopNews__BBView)

④ Associate the backbone view with the Handlebars template

⑤ Variable for the selector where the rendered Handlebars template will be mounted.

⑥ `renderDesktop` and `renderPhone` are Cloudcraze helper functions that are executed when the Backbone view is rendered. depending on the type of design (responsive/boot3 uses renderDesktop, while adaptive/classic uses both depending on the screen width).

⑦ Sets the selector where the rendered Handlebars template will be mounted.

⑧ REST web service endpoint to call (GET)

⑨ Sets the Handlebars contents to use the data retrieved from the web service call

⑩ Initializes and render the TopNews_BBView Backbone view.

If you are struggling to understand how the various front-end libraries fit together to render the final DOM consumed by the browser,

1. Refer to these diagrams in the Appendix.
2. Try setting some breakpoints through your browser's Developer Tools to inspect various Handlebars Helpers and specific Backbone View/Model functions to understand timing and scope of when these different parts of the code are called.

c. Save your changes (On the Developer Console menu, click: **File → Save**)

4. Configure the Page Include to be used on the Home Page:

a. In your Salesforce Org, click **CC Admin** from the main tab.

b. On the right side dropdown select the **DefaultStore** Storefront.

c. On the left side bar, near the top, under "Settings" section, click on **Configuration Settings**.

d. Click on **New** button to create a new Configuration Setting and enter the following to **enable** the page include:

When filling out the value for the `Page` input field referenced below, you should keep the following in mind. Start out by typing the name of the page of interest. E.g Type `Home` and then select the appropriate value from the autocompleted suggestions in the input field.

Module

Body Includes End

Configuration

Enabled

Page

Home (HP)

Value

TRUE

a. Next, click on **New** button to create a new Configuration Setting and enter the following to **set** the page include (don't forget the `c__` prefix before the name of your VF page):

Module

Body Includes End

Configuration

Page Include Name

Page

Home (HP)

Value

c__ccTraining_PI_TopNews

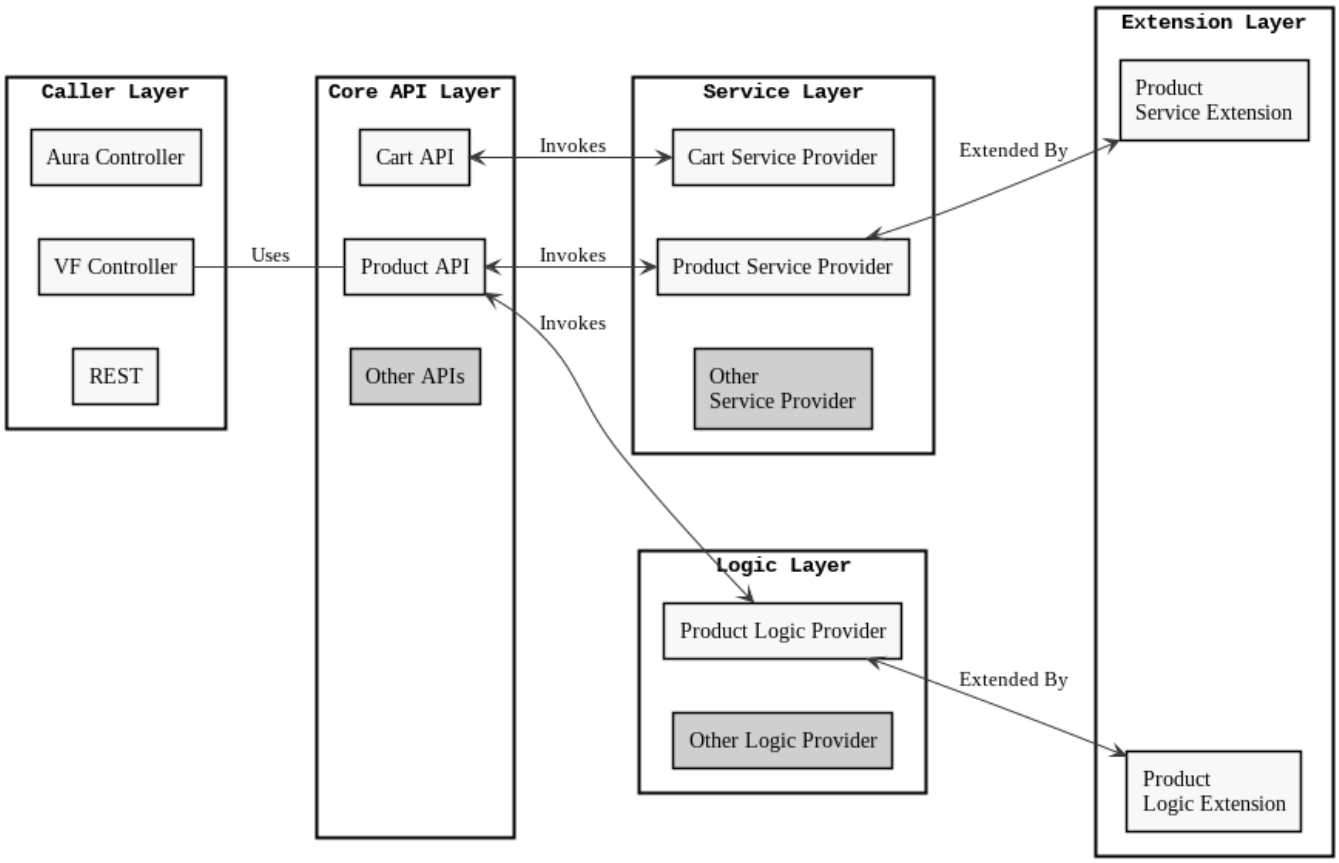
5. Create and Activate a new Configuration Cache:
   a. On the right side dropdown, click on the **Global Settings** link.
   b. On the left side bar, select **Configuration Cache Management**.
   c. Click on **Build New**. After a few seconds, click on **Refresh List**. A new configuration cache should have been built. Click on **"Activate"** to enable it.
6. Test your changes:
   a. In your SalesForce Org, click on **Contacts** from the main tab bar.
   b. Search for **Jon Amos**. Click on the name to open the contact info.
   c. On the Contact Detail Section, click on **Manage External User**, and from the dropdown, select "Log in to Community as User".
   d. Open the home page and confirm that the News widget shows up at the bottom.

Bonus Points:

1. Improve the look and feel by manipulating the HTML and the Theme's CSS.
2. Make the content configurable, by using either CC Content or CC Page Labels.
3. Move the web service call logic into a custom Backbone Model.
4. Call other webservices and webservices verbs (e.g. POST, PUT, etc.).
5. Add a controller for custom logic.
6. Use remote actions.

Chapter 3. Backend

3.1. BACK-END OVERVIEW



There are primarily two ways of extending the back-end functionality in CloudCraze.

1. Overriding the API Data Service and Logic Providers
2. Overriding the Global Extension Points
    a. Overriding the back-end means extending an Apex Class from the managed package, and configuring CloudCraze to use this custom Apex Class

3.2. Global APIS

The CloudCraze global APIs provide access to CloudCraze business functionality that cannot be accessed either by extending the user interface, querying, or updating the CloudCraze custom Salesforce objects. Each global API has been segmented to cover specific areas of functionality in a logical grouping

Listed below are some of the Global APIs

- ccApi
- ccApiAccount

- ccApiAddress
- ccApiAddressBook
- ccApiCart
- ccApiCategory
- ccApiConfig
- ccApiContact
- ccApiCoupon
- ccApiI18N
- ccApiInvoice
- ccApiOrder
- ccApiPrivateCache
- ccApiProduct
- ccApiPromotion
- ccApiPublicCache
- ccApiRelatedProduct
- ccApiSeller
- ccApiSpec
- ccApiStorefront
- ccApiSubscPage
- ccApiSubscription
- ccApiTerms
- ccApiPriceList
- ccApiStoredPayment
- ccApiSubProdTerm
- ccApiTransactionPayment

An up-to-date list of the all the Global APIs can be found here on the wiki.

3.2.1. Sizing

Field Sizing

- Small – ccAPI.SZ_S
- Medium – ccAPI.SZ_M
- Large – ccAPI.SZ_L (Default)
- X-Large – ccAPI.SZ_XL

Related Query Sizing

ccAPI.SZ_REL

Associated API Calls

Sometimes it is not desirable to query any other API's. E.g. we just want Cart data without any Address or Product data. To this end, we can specify that an API request shouldn't propagate further requests to other APIs or Entities. **ccAPI.SZ_ASSC** is the Boolean flag that shuts off further API queries.

Return Format

API calls, by default, return maps of string objects (Map<String, Object>) where the object data is hashed under a return key. The object data, by default, is returned as a list of Map<String, Object> where the fields names form the key. Internally, sometimes this behavior is undesirable and the caller would like to work directly with sObject types. To that end, the APIs can return a list of sObjects. Since it is not always possible to stitch the data back together to create a consolidated sObject, the data is returned in separate strings, defined by each API. **ccAPI.SZ_SKIPTRZ** is the Boolean flag that allow data to come back as an sObject.

Refetch Data

Some API's support an implicit refetch of the data to save the caller to do a fetch directly after, say, a create. APIs will mark which services and calls support the refetch mechanism. **ccAPI.SZ_REFETCH** is the Boolean flag that signifies the caller wants to use the implicit refetch mechanism.

3.2.2. Version

The Global APIs have various versions. CC 4.7 is currently on version 6 of the APIs.

Invoking an API with its current version or perhaps a specific version can be done as shown below:

```
ccrz.ccApi.API_VERSION ⇒ ccAPI.CURRENTVERSION
```

Or for a specific version

```
ccrz.ccApi.API_VERSION ⇒ 4
```

The recommended approach is to use a Global API instead of a SOQL query when retrieving records for custom objects that are part of the managed package.

```
public with sharing  class ccTrainingAPIExample
{
  public ccTrainingAPIExample()
  {

  }

    public static void sampleMethod()
    {
     Set<String> productIdList = new Set<String>{'ID_1', 'ID_2'};

    Map<String, Object> inputData = new Map<String, Object>{
        ccrz.ccAPIProduct.PRODUCTIDLIST => productIdList,
        ccrz.ccAPI.API_VERSION => ccrz.ccAPI.CURRENTVERSION
    };

    try
    {
        Map<String, Object> outputData = ccrz.ccAPIProduct.fetch(inputData);

        if (outputData.get(ccrz.ccAPIProduct.PRODUCTLIST) != null)
        {
            // The cast to List<Map<String, Object>> is necessary...
            List<Map<String, Object>> outputProductList = (List<Map<String, Object>>)
outputData.get(ccrz.ccAPIProduct.PRODUCTLIST);

            // The cast, again, is necessary...
            String productName = (String) outputProductList[0].get('sfdcName');
        }
    }
    catch (Exception e)
    {
        // Error handling...
    }
  }
}
```

3.3. Data Service Provider

For every Global API, there is a corresponding Data Service Provider. The Data Service Providers usually fetches the data used by downstream logic classes in the CloudCraze platform.

getFieldsMap

The select clause in the query

getSubQueryMap

Configuring related queries

getDirectQueryMap

Configuring direct queries

getFilterMap

Configuring the where clause in the query

getOrderByMap

Set order for the query

buildQuery

pulls all methods together

3.3.1. Exercise: Modifying the Product Data Service Provider

In this exercise, we'll show you how to modify the data service provider for CC Product and have it fetch data from a custom field.

Use Case

1. Extend the Data Service Provider class for the CC Product Custom object
2. Override the `getFieldsMap` method and fetch an additional attribute for the product object
   a. The attribute will be added to the to the `OBJECTFIELDS` constant in CloudCraze
3. Configure CloudCraze to use the new data service provider

You must create the Custom Field of type Text on Product with the API name `Brand__c`. See below for further instructions.

a. **Add a new custom field to the CC product object**

   1. Create a new attribute for brand on the product object. This will be used to reference the product's brand.
   2. While in the setup menu, Type `Object` in the quick find box.
   3. Select **Objects** under the **Build → Create → Objects** menu.
   4. Select **CC Product** from the list of the available custom objects. **Do not** select the Edit button.
   5. Scroll down to the `Custom Fields and Relationships` section and then select `New`
   6. Select `Text` as the field type on the next screen
   7. Set the following on the subsequent screen.
      i. Field Label: Brand
      ii. Length: 255
      iii. Field Name: Brand
   8. Make the field visible for all the profiles.
   9. Add the `Brand` attribute to the layout for `CC Product` and save your changes.

b. **Implement your new data service provider for Products**

   1. Create a new Apex class called `ccTrainingProductService` that extends `ccrz.ccServiceProduct`

2. Override the `getFieldsMap` method.
3. Inside the `getFieldsMap` override, retreive the base fields this class returns by default
4. While still inside this method, concatenate `Brand__c` to the String that contains the default base fields
5. Return the newly created String

```
global with sharing  class ccTrainingProductService extends ccrz.ccServiceProduct ①
{

    global virtual override Map<String, Object> getFieldsMap(Map<String, Object> inputData) ②
    {
        inputData = super.getFieldsMap(inputData); ③

        String objectFields = (String)inputData.get(ccrz.ccService.OBJECTFIELDS); ③

        objectFields += ',Brand__c' ; ④

        return new Map <String,Object> {ccrz.ccService.OBJECTFIELDS => objectFields}; ⑤
    }

}
```

c. **Configure CloudCraze to use your new data service Provider**

1. In your Salesforce Org, click **CC Admin** from the main tab.
2. On the right side dropdown select the **DefaultStore** Storefront.
3. On the left side bar, select "Service Management"
4. Scroll down to the data service provider section and search for ccServiceProduct
5. Set the class name to `c.ccTrainingProductService`

d. **Expose the new field on the Product Detail Page**

1. Open the SalesForce Developer Console:
    i. In your SalesForce Org, click the **Developer Console** link under the dropdown menu **under your name**.
    ii. A new window should pop-up.
    iii. Alternatively, you can use this URL: https://<YOUR_SALESFORCE_DOMAIN>/_ui/common/apex/debug/ApexCSIPage
2. Create a New VisualForce page named **ccTrainingProductDetailBIE**
    i. In the Developer Console menu, select: **File → New → VisualForce Page**
    ii. In the popup window (New Apex Page), enter **"ccTrainingProductDetailBIE"** as your page name and click **ok**.
    iii. Add the code snippet below to your new page : please note that the reference code was added for your convenience. It is also available on the wiki here
    iv. Conversely, you can also access the code by copying over the source content via your browser menu

```
<apex:page docType="html-5.0" sidebar="false" showHeader="false"
standardStylesheets="false" applyHtmlTag="false">

    <script>
        CCRZ.uiProperties.productDetailView.desktop.tmpl = 'CCTrainingProductDetail-
Desktop'; ①
    </script>

 <script id="CCTrainingProductDetail-Desktop" type="text/template"> ②
   <div class="panel panel-default product_detail_container product_type_standard
phoneProductItem cc_panel cc_product_detail_container cc_product_type_standard">
      <div class="panel-heading cc_heading">
         <h3 class="panel-title cc_title">{{this.product.prodBean.name}}</h3>
      </div>
      <div class="panel-body product_detail cc_body cc_product_detail" >
         <div class="messagingSection-Error" role="alert" style="display: none"></div>
         <div class="messagingAction-Info" role="alert" style="display: none">
            <button type="button" class="close cc_close" data-dismiss="alert" aria-
label="Close"><span aria-hidden="true">&times;</span></button>
         </div>
         <div class="row">
            <div class="col-md-5">
               <div class="prod_media cc_prod_media">
                  {{#if this.mediaWrappers.[SVG Interactive Diagram]}}
                     {{#each this.mediaWrappers.[SVG Interactive Diagram]}}
                        <div class="interactive cc_interactive">{{displaySVG this
'mainProdImage prodDetail img-responsive'}}</div>
                     {{/each}}
                  {{/if}}
                  <div id="altImageModal" class="modal fade cc_alt_image_modal" tabindex="-
1" role="dialog">
                     <div class="modal-dialog cc_modal-dialg" role="document">
                        <div class="modal-content cc_modal-content">
                           <div class="modal-header cc_modal-header">
                              <button type="button" class="close cc_close" data-
dismiss="modal" aria-label="Close"><span aria-hidden="true">&times;</span></button>
                              <h4 class="modal-title cc_modal-title"
id="myModalLabel"></h4>
                           </div>
                           <div class="modal-body cc_modal-body">
                              <img class="modalImg cc_modal_img" src=""/>
                              <p class="modalText cc_modal_text"></p>
                           </div>
                        </div>
                     </div>
                  </div>
                  {{#if this.product.mediaWrapper }}
                     <div class="cc_product_detail_photo_container" id="photoContainer">
                        <div class="row">
                           <div class="col-md-12">
                              <div class="cc_main_prod_image">{{displayImage
this.product.mediaWrapper 'mainProdImage prodDetail img-responsive'}}</div>
                           </div>
                        </div>
                        {{#if this.mediaWrappers.[Alternate Images]}}
                           <div class="row">
                              {{#each this.mediaWrappers.[Product Image]}}
                                 <div class="col-xs-3">
```

```
                                    <img class="alternate cc_alternate img-responsive
thumbnail" src="{{productImage this}}" data-id="{{this.uri}}"/>
                                 </div>
                              {{/each}}
                              {{#each this.mediaWrappers.[Alternate Images]}}
                                 <div class="col-xs-3">
                                    <img class="alternate cc_alternate img-responsive
thumbnail" src="{{productImage this}}" data-id="{{this.uri}}"/>
                                 </div>
                              {{/each}}
                           </div>
                        {{/if}}
                     </div>
                  {{else}}
                     {{#if this.mediaWrappers.[SVG Interactive Diagram]}}
                     {{else}}
                        <div class="cc_product_detail_photo_container" id="photoContainer">
                           <div class="row">
                              <div class="col-md-12">
                                 <div class="cc_main_prod_image img-
responsive">{{displayImage this.product.mediaWrapper 'mainProdImage prodDetail'}}</div>
                              </div>
                           </div>
                        </div>
                     {{/if}}
                  {{/if}}
               </div>
            </div>
            <div class="col-md-7">
               <div class="product_detail_item wishFinder cc_wish_finder" data-
sku="{{this.product.prodBean.sku}}">

                  <h4 class="product_title
cc_product_title">{{this.product.prodBean.name}}</h4>
                  {{#ifDisplay 'PD.DsplSku'}}
                     <div class="sku cc_sku">
                        <span class="cc_label">{{pageLabelMap
'ProductDetailsInc_SKU'}}</span>
                        <span class="value cc_value">{{this.product.prodBean.sku}}</span>
                     </div>
                  {{/ifDisplay}}
                  {{#ifDisplay 'PD.DsplUOM'}}
                     <div class="uom cc_uom">
                        <span class="cc_label">{{pageLabelMap
'ProductDetails_UnitOfMeasure'}}</span>
                        <span class="value cc_value">{{pageLabelPrefixMap 'UOM_'
this.product.prodBean.UnitOfMeasure}}</span>
                     </div>
                  {{/ifDisplay}}
                  {{#ifDisplay 'PD.DsplAvlb'}}
                     <div class="inventory cc_inventory">
                        <span class="cc_label">{{pageLabelMap
'ProductDetails_Availability'}}</span>
                        {{#ifStoreSetting 'InventoryCheckFlag__c'}}
                           <span class="value cc_value">{{{this.product.inventory}}}</span>
                        {{else}}
                           <span class="value cc_value">{{{pageLabelMap
this.product.availMsg}}}</span>
                        {{/ifStoreSetting}}
```

```
                    </div>
                {{/ifDisplay}}
                <!-- TODO Exercise-->
                <div class="brand">
                      <span class="cc_label">{{pageLabelMap
'CCTrainingProductDetails_Brand'}}</span>
                       ③
                </div>
                {{#ifDisplay 'PR.Enabled'}}
                   <div id="avgRating" class="rateit cc_rateit" data-rateit-
value="{{this.product.avgRating}}" data-rateit-ispreset="true" data-rateit-readonly="true">
                      <span class="cc_label">{{pageLabelMap 'NumberofReviews'
this.numberOfReviews }}</span>
                   </div>
                {{/ifDisplay}}
                {{#ifDisplay 'PD.DsplSDesc'}}
                   <div class="shortDesc cc_short_desc">
                      <p class="pblock
cc_pblock">{{{this.product.prodBean.shortDesc}}}</p>
                   </div>
                {{/ifDisplay}}
                <hr>
                <div class="row">
                   <div class="col-md-12">
                      {{#if this.product.showPricing}}
                         {{#ifDisplay 'PD.DsplPrc'}}
                            {{#if this.product.price}}
                               <div class="price_block cc_price_block">
                                  {{#ifDisplay 'PD.DsplListPrc'}}
                                     {{#if this.product.basePrice}}
                                        <p class="baseprice cc_baseprice">
                                           <span class="cc_label">{{pageLabelMap
'ProductDetails_ListPrice'}}</span>
                                           <span class="value cc_value">{{price
this.product.basePrice}}</span>
                                        </p>
                                     {{/if}}
                                  {{/ifDisplay}}
                                  <p class="price cc_price">
                                     <span class="cc_label">{{pageLabelMap
'Price'}}</span>
                                     {{#if this.highAttrPrice}}
                                                                        {{price
this.lowAttrPrice}} - {{price this.highAttrPrice}}
                                                                    {{else}}
                                                                        {{price
this.product.price}}
                                                                    {{/if}}
                                                                    </span>
                                  </p>
                                  <p class="cc_sold_by">
                                     {{#if this.product.sellerID}}
                                        <span class="soldbylabel
cc_sold_by_label">{{pageLabelMap 'Prod_SoldBy'}}</span>
                                        <span class="soldbyname cc_sold_by_name">{{pdp-
seller-field 'sfdcName' this.product.sellerID this.sellers}}</span>
                                     {{/if}}
                                  </p>
                                  {{#ifDisplay 'PD.DsplSvPrc'}}
```

```
                                    {{#if this.product.savings}}
                                       <p class="savings">
                                          <span class="cc_label">{{pageLabelMap
'YouSave'}}</span>
                                          <span class="value cc_value">{{price
this.product.savings}}</span>
                                       </p>
                                    {{/if}}
                                 {{/ifDisplay}}
                              </div>
                              <hr>
                           {{/if}}
                        {{/ifDisplay}}
                     {{/if}}
                  </div>
               </div>
               {{#if this.product.canAddtoCart}}
{{#unless this.primaryAttr}}
                  <div class="quantity_block gp_quantity_block cc_quantity_block">
                     {{#if this.product.qtySkipIncrement}}
                        <div class="row cc_qty_control_row">
                           <div class="col-md-10 col-md-offset-2">
                              <div class="form-group">
                                 <div class="input-group cc_input_group">
                                          <span class="input-group-btn
cc_input_group_btn">
                                             <input type="button" value="{{pageLabelMap
'Prod_QtyDecrFF'}}" class="btn btn-default btn-sm minusFF cc_minusff">
                                             <input type="button" value="{{pageLabelMap
'Prod_QtyDecrSingle'}}" class="btn btn-default btn-sm minus cc_minus">
                                          </span>
                                    <input type="text" readonly="true" name="qty"
value="0" class="entry form-control input-sm cc_entry" maxlength="7" />
                                    <span class="input-group-btn cc_input_group_btn">
                                          <input type="button" value="{{pageLabelMap
'Prod_QtyIncrSingle'}}" class="btn btn-default btn-sm plus cc_plus">
                                             <input type="button" value="{{pageLabelMap
'Prod_QtyIncrFF'}}" class="btn btn-default btn-sm plusFF cc_plusff">
                                          </span>
                                 </div>
                              </div>
                           </div>
                        </div>
                     {{else}}
                        {{#ifEquals this.product.qtyIncrement 1}}
                           <div class="row cc_qty_control_row">
                              <div class="col-md-6 col-md-offset-6">
                                 <div class="form-group">
                                    <div class="input-group cc_input_group">
                                             <span class="input-group-btn
cc_input_group_btn">
                                                <input type="button"
value="{{pageLabelMap 'Prod_QtyDecrSingle'}}" class="btn btn-default btn-sm minus
cc_minus">
                                             </span>
                                       <input id="qty{{index}}"
name="quickadd[{{index}}].qty" value="0" class="qty entry form-control input-sm cc_entry"
maxlength="7" />
                                       <span class="input-group-btn cc_input_group_btn">
```

```
                                                <input type="button"
value="{{pageLabelMap 'Prod_QtyIncrSingle'}}" class="btn btn-default btn-sm plus cc_plus">
                                                </span>
                                             </div>
                                          </div>
                                       </div>
                                    </div>
                                 {{else}}
                                    <div class="row cc_qty_control_row">
                                       <div class="col-md-12">
                                          <div class="form-horizontal">
                                             <div class="form-group">
                                                <label for="qty" class="col-sm-7 control-label
cc_qty">{{pageLabelMap 'Qty'}}</label>
                                                <div class="col-sm-5">
                                                   <input type="text" id="qty" name="qty"
value="1" class="input-text entry plus_minus cc_entry form-control" maxlength="7" />
                                                </div>
                                             </div>
                                          </div>
                                       </div>
                                    </div>
                                 {{/ifEquals}}
                              {{/if}}
                              <input type="hidden" name="qtyIncrement"
value="{{this.product.qtySingleIncrement}}" class="item_qtyIncrement cc_item_qty_increment"
/>
                              <input type="hidden" name="qtySkipIncrement"
value="{{this.product.qtySkipIncrement}}" class="item_qtySkipIncrement
cc_item_qty_skip_increment" />
                           </div>
                           {{#unless this.showNewSubscriptionSelection}}
                              {{#if this.product.prodBean.showSubscriptionSelection}}
                                 <div class="row">
                                    <div class="col-md-12">
                                       <div class="cc_subscription_selection_div">
                                          <p class="subscription_selection
cc_subscription_selection">
                                             <span class="subscriptionLabel
cc_subscription_label">{{pageLabelMap 'Subscribe_And_Save_Label'}}</span>
                                             <select class="subscriptionFrequencySelection
cc_subscription_frequency_selection" data-subscription="{{this.product.prodBean.sku}}">
                                                {{#each
this.product.prodBean.subscriptionFrequencies}}
                                                <option value="{{safeQuote
this}}">{{pageLabelMapMultiString 'Subscribe_And_Save_' this}}</option>
                                                {{/each}}
                                             </select>
                                          </p>
                                       </div>
                                    </div>
                                 </div>
                              {{/if}}
                              <div class="row">
                                 <div class="col-md-10 col-md-offset-2">
                                    <div class="wishButtons plus_minus cc_plus_minus pull-
right"></div>
                                 </div>
                              </div>
```

```
                        <div class="row">
                           <div class="col-md-8 col-md-offset-4">
                              <div class="action cc_action">
                                 <button type="button" class="btn btn-primary btn-sm
addItem cc_add_item pull-right" data-sku="{{this.product.prodBean.sku}}" data-
seller="{{this.product.sellerID}}">{{pageLabelMap
'Component_MiniwishList_AddToCart'}}</button>
                              </div>
                           </div>
                        </div>
                     {{/unless}}
{{/unless}}
                  {{/if}}
               </div>
            </div>
         </div>
{{#if this.primaryAttr}}
                        <div class="row">
                            <div class="col-md-12">
                                <div class="cc_product_attributes"></div>
                            </div>
                        </div>
                        <div class="row">
                            <div class="col-md-12">
                                <div class="cc_product_attributes_batch_header">
                                </div>
                            </div>
                        </div>
                     {{/if}}
         <div class="row">
            <div class="col-md-12">
                <div class="cc_product_attributes"></div>
            </div>
        </div>
        <div class="row">
            <div class="col-md-12">
                <div class="cc_product_attributes_batch_header">
                </div>
            </div>
        </div>
        <div class="row">
            <div class="col-md-7 col-md-offset-5">
               {{#if this.showNewSubscriptionSelection}}
                  {{#if this.product.canAddtoCart}}
                     <div class="cc_subscription_selection_div">
                        <h4>{{pageLabelMap 'Subscribe_And_Save_Label'}}</h4>
                        {{#each this.subProdTerms}}
                           {{#if this.CC_NO_SUBSCRIPTION}}
                              <div class="cc_item_container">
                                 <div class="radio cc_radio">
                                    <label class="cc_radio_name">
                                       <input value="nosuboption" data-nme="nosuboption"
type="radio" name="subOptionGuider{{this.productSKU}}" {{this.checked}}/>
                                       {{pageLabelMap 'Prod_NoSubscriptionOption' (price
this.productPrice)}}</label>
                                 </div>
                              </div>
                           {{else}}
                              <div class="cc_item_container">
```

```
                                <div class="radio cc_radio">
                                   <label class="cc_radio_name">
                                      <input value="{{this.sfid}}" data-
nme="{{this.name}}" type="radio" name="subOptionGuider{{this.productSKU}}"
{{this.checked}}/>
                                      {{#if this.modifierSubscriptions}}
                                         {{insertTokens this.pdpDisplayName
this.displayName (price this.productPrice) (price this.subscriptionPrice)
(pageLabelPrefixMap 'Subscriptions_Frequency_' this.orderFrequencyUOM) (pageLabelPrefixMap
'Subscriptions_Frequency_' this.installmentFrequencyUOM) this.orderFrequency
this.installmentFrequency this.installmentCount this.orderCount (price
this.modifierSubscriptions.[0].productPrice) (price
this.modifierSubscriptions.[0].subscriptionPrice) this.modifierSubscriptions.[0].orderCount
this.modifierSubscriptions.[0].installmentCount}}
                                      {{else}}
                                         {{insertTokens this.pdpDisplayName
this.displayName (price this.productPrice) (price this.subscriptionPrice)
(pageLabelPrefixMap 'Subscriptions_Frequency_' this.orderFrequencyUOM) (pageLabelPrefixMap
'Subscriptions_Frequency_' this.installmentFrequencyUOM) this.orderFrequency
this.installmentFrequency this.installmentCount this.orderCount}}
                                      {{/if}}
                                   </label>
                                </div>
                             </div>
                          {{/if}}
                       {{/each}}
                       <div class="action pull-right cc_action" >
                          <button type="button" class="btn btn-primary addItem
cc_add_item" data-sku="{{this.product.prodBean.sku}}">{{pageLabelMap
'Component_MiniwishList_AddToCart'}}</button>
                       </div>
                    </div>
                 {{/if}}
              {{/if}}
           </div>
        </div>
     </div>
  </div>
  <div class="tabSection"></div>
  <div class="widgetSection"></div>
</script>


</apex:page>
```

① Handlebars template override, this template override can also be done in the uiproperties.js file.

② Handlebars template ID defined here should always match the value in the override from the previous step.

③ Placeholder for the brand attribute needs to be displayed. Copy and paste `<span class="value cc_value">{{{pageLabelMap this.product.prodBean.brand}}}</span>` into the annotated section in the reference code.

3. Configure the Page Include to be used on the Product Detail Page:

A. In your Salesforce Org, click **CC Admin** from the main tab.

B. On the right side dropdown select the **DefaultStore** Storefront.

C. On the left side bar, near the top, under "Settings" section, click on **Configuration Settings**.

When filling out the value for the `Page` input field referenced below, you should keep the following in mind. Start out by typing the name of the page of interest. E.g Type `Product Details` and then select the appropriate value from the autocompleted suggestions in the input field.

D. Select **New** to create a Configuration Setting and enter the following to **enable** the page include

Module

Body Includes End

Configuration

Enabled

Page

Product Details (PDP)

Value

TRUE

E. Select **New** to create a Configuration Setting and enter the following to **set** the page include (don't forget the `c__` prefix before the name of your VF page):

Module

Body Includes End

Configuration

Page Include Name

Page

Product Details (PDP)

Value

c__ccTrainingProductDetailBIE

F. Create a page label for the field that exposes the value of the brand attribute

Page Name

ProductDetails

Page Label Name

CCTrainingProductDetails_Brand

Value

Brand

Storefront

DefaultStore

G. Create and Activate a new Configuration Cache:

H. On the right side dropdown, click on the **Global Settings** link.

I. On the left side bar, select **Configuration Cache Management**.

J. Click on **Build New**. After a few seconds, click on **Refresh List**. A new configuration cache should have been built. Click on **"Activate"** to enable it.

e. **Verify your changes**

1. Navigate to the **CC Product** tab in your Salesforce org.
2. Edit any product of your choosing by setting a value in its **Brand** attribute (e.g I love CloudCraze).
3. In your browser, verify that the brand attribute is now getting populated in the productDetailModel.
   i. Search for the product you edited in yor storefront and navigate to the Product Detail page.
   ii. Open your browser's Javascript console and then type **CCRZ** and hit enter.
   iii. A shortcut for accessing the Javascript console in Chrome is by typing **Command + option + J** if you are a MAC user.
   iv. Conversely, you can access this by navigating to **View → Developer → Javascript Console** in Chrome.
4. Navigate through the tree structure of the CCRZ object to view the contents of the `productDetailModel`
5. You can also just type `CCRZ.productDetailModel.attributes.product.prodBean` and observe that brand is now automatically populated.
6. Finally, verify that the new attribute is visible on the product detail page.

3.4. Logic Service Provider

The Logic service layer provides a way of encapsulating specific business logic into discrete blocks that can be extended and modified by subscribers. Logic services are named according to the top-level entitiy they're associated with and the business logic it provides.

Version specific logic classes will reference the version that they are linked to. e.g.
`ccLogic<ENTITY><FUNCTION><VERSION> → ccLogicProductPricing6`

Examples of Logic Providers

- ccLogicAccountCreateAccount
- ccLogicAccountGetAnonymous
- ccLogicAccountGetCurrent

- ccLogicAccountValidateNew
- ccLogicAddressBookCreateAddressBook
- ccLogicAddressBookRemoveAddressBook
- ccLogicAddressCreateAddress
- ccLogicCartAddTo
- ccLogicCartPrice
- ccLogicCartAdjustment
- ccLogicCartClone
- ccLogicCartCreate

A complete list of the all the logic providers can be found in the service management section in CC Admin.

3.4.1. Exercise: Override the Add To Cart Logic Provider

In this exercise, we'll show you how to extend the logic service provider for the add to cart process. You'll primarily allow a product to be added to the cart or not based on certain business constraints.

Use Case

1. Create a new attribute with an API name of `RestrictedMaterial__c` on the account object. This will reference the material an account is restricted from purchasing.
2. Update the data service providers for Account objects to include this new field. You should only create this class if you haven't done so in a previous exercise.
3. Configure CloudCraze to use your data service provider as needed.
4. Create a new attribute with an API name of `Material__c` on the product object. This will reference the material used for the product.
5. Update the data service providers for the Product object to include this new field. You should create this class data if you haven't done so in a previous exercise.
6. Configure CloudCraze to use your data service provider as needed.
7. Override the Logic Provider class for the add-to-cart functionality.
8. Inside the overriden method, determine if a product can be added to the cart based on its material.
9. Configure CloudCraze to use the new logic provider and data service providers as needed.

Prior to this exercise, you must create an extension for the data service providers for CC Product and Accounts. See below for further instructions.

a. **Add a custom field to the Account object**

1. Create a new attribute for restricted material on the account object. This will reference the material an account is restricted from purchasing.

2. While in the setup menu, Type `Account` in the quick find box.
3. Select **Fields** under the **Build → Customize → Account → Fields** menu.
4. Scroll down to the `Custom Fields and Relationships` section and then select `New`
5. Add a picklist of restricted materials with the following details
    i. Field Label: Restricted Material
    ii. Values: Metal, Aluminum and Titanium
    iii. Ensure you enter each of these values on a separate line.
6. Ensure that the auto-generated field name is `RestrictedMaterial` and not `Restricted_Material`
7. Make the field visible for all the profiles.
8. Add the `Restricted Material` attribute to the Account layout and save your changes.
9. Keep advancing through the setup screens and then save your changes.

b. **Implement the data service provider for Accounts**

1. Create a new Apex class called `ccTrainingServiceAccount` that extends `ccrz.ccServiceAccount`
2. Override the `getFieldsMap` method.
3. Inside the `getFieldsMap` method, retrieve the base fields this class returns by default.
4. While still inside this method, concatenate `RestrictedMaterial__c` to the String that contains the default base fields
5. Return the newly created String

```
global with sharing class ccTrainingServiceAccount extends ccrz.ccServiceAccount ①
{
    global virtual override Map<String, Object> getFieldsMap(Map<String, Object> inputData) ②
    {
        inputData = super.getFieldsMap(inputData); ③

        String objectFields = (String)inputData.get(ccrz.ccService.OBJECTFIELDS); ③

        objectFields += ',RestrictedMaterial__c' ; ④

        return new Map <String,Object> {ccrz.ccService.OBJECTFIELDS => objectFields}; ⑤
    }
}
```

c. **Configure CloudCraze to use your new data service Provider**

1. In your Salesforce Org, click **CC Admin** from the main tab.
2. On the right side dropdown, select the **DefaultStore** Storefront.
3. On the left side bar, select `Service Management`
4. Scroll down to the Logic provider section and search for **ccServiceAccount**
5. Set the class name to `c.ccTrainingServiceAccount`

6. Save your changes

d. **Add a custom field to the CC Product object**

1. Create a new attribute with an API Name of `Material__c` on the `CC Product` object. This will be used to reference the product's material.
2. While in the setup menu, Type `Object` in the quick find box.
3. Select **Objects** under the **Build → Create → Objects** menu.
4. Select **CC Product** from the list of the available custom objects. **Do not** select the Edit button.
5. Scroll down to the `Custom Fields and Relationships` section and then select `New`
6. Select `Picklist` as the field type on the next screen
7. Add a picklist of materials with the following details
    i. Field Label: Material
    ii. Values: Metal, Aluminum and Titanium
    iii. Ensure you enter each of these values on a separate line.
8. Verify that the auto-generated field name is `Material`
9. Make the field visible for all the profiles.
10. Add the `Material` attribute to the layout for `CC Product` and save your changes.
11. Keep advancing through the setup screens and then save your changes.

e. **Implement the data service provider for CC Product**

1. Create a new Apex class called `ccTrainingProductService` that extends `ccrz.ccServiceProduct`
    i. Ignore this step if you've created this class in a previous exercise
2. Override the `getFieldsMap` method.
    i. Ignore this step if this has been done in a previous exercise
3. Inside the `getFieldsMap` method, retrieve the base fields this class returns by default.
4. While still inside this method, concatenate `Material__c` to the String that contains the default base fields
5. Return the newly created String

```
global with sharing  class ccTrainingProductService extends ccrz.ccServiceProduct ①
{
    global virtual override Map<String, Object> getFieldsMap(Map<String, Object> inputData) ②
    {
        inputData = super.getFieldsMap(inputData); ③

        String objectFields = (String)inputData.get(ccrz.ccService.OBJECTFIELDS);  ③

        objectFields += ',Brand__c,Material__c' ;  ④

        return new Map <String,Object> {ccrz.ccService.OBJECTFIELDS => objectFields};  ⑤
    }
}
```

f. **Configure CloudCraze to use your new data service provider if it's not already in place**

1. In your Salesforce Org, click **CC Admin** from the main tab.
2. On the right side dropdown select the **DefaultStore** Storefront.
3. On the left side bar, select `Service Management`
4. Scroll down to the Logic provider section and search for ccServiceProduct
5. Set the class name to `c.ccTrainingProductService`

g. **Update the restricted material on the account record associated with your user**

1. Go to the Account tab in your Salesforce org and edit the account record linked to your user. Add a new Restricted Material `(e.g. Metal)`, and save your changes.
2. Update the material on the product record we'll be using for the exercise - You can pick any product of your choosing.
3. Go to the CC Product tab in Salesforce, search for the product you are interested in and set its material to the same value as that referenced in your user's account (e.g. Metal).
4. Ensure you are using a product the user is entitled to in the storefront.
5. Save your changes.

h. **Implement your logic provider**

1. Create a new Apex class called `ccTrainingLogicCartAddTo` that extends `ccrz.ccLogicCartAddTo`
2. Override the `processInputData` method.
3. Inside the `processInputData` override, ensure we are not trying to add a coupon to the cart.
4. While still in the `processInputData` method, fetch the product data and retrieve the value saved in the attribute named `material`
5. If the product has a material specified for it, proceed to fetch the account data of the user.
6. Fetch the account data and retrieve the value saved in the attribute named `restrictedMaterial`
7. If the `material` attribute on the product matches the `restrictedMaterial` on the account, then do not add the product to the cart.

```
```

```
global with sharing class ccTrainingLogicCartAddTo extends ccrz.ccLogicCartAddTo ①
{
   global virtual override Map<String,Object> processInputData(Map<String,Object> inputData) ②
   {
        Boolean restrictedProduct = false;

        String couponCode = (String)inputData.get(ccrz.ccAPICart.COUPON_CODE);

        if(!ccrz.ccUtil.isKeyValued(inputData, ccrz.ccApiCart.COUPON_CODE)) ③
        {
           List<Object> incomingLineData = (List<Object>) inputData.get(ccrz.ccApiCart.LINE_DATA);

            Set<String> productSkuSet = new Set<String>();

            Set<String> productIdSet = new Set<String>();

            for(Object lineData : incomingLineData)
            {
                Map<String, Object> castlineData = (Map<String, Object>) lineData;

                String prodId = (String)castlineData.get('productId');
                String prodSku = (String)castlineData.get('sku');

                if(!String.isEmpty(prodId))
                {
                    productIdSet.add(prodId);
                }
                else if(!String.isEmpty(prodSku))
                {
                    productSkuSet.add(prodSku);
                }
            }

            Map<String, Object> inputMap = new Map<String, Object>{
                ccrz.ccApi.API_VERSION => ccrz.ccApi.CURRENT_VERSION,
                ccrz.ccAPI.SIZING => new Map<String, Object>{
                    ccrz.ccAPIProduct.ENTITYNAME => new Map<String, Object>{
                            ccrz.ccAPI.SZ_DATA => ccrz.ccAPI.SZ_XL
                    }
                }
            };

            if(productIdSet.size() > 0)
            {
                inputMap.put(ccrz.ccApiProduct.PRODUCTIDLIST, productIdSet);
            }
            else if(productSkuSet.size() > 0)
            {
              inputMap.put(ccrz.ccApiProduct.PRODUCTSKULIST, productSkuSet);
            }

            Map<String, Object> productData = ccrz.ccAPIProduct.fetch(inputMap); ④

            List<Map<String, Object>> outputProductList = (List<Map<String, Object>>)
productData.get(ccrz.ccAPIProduct.PRODUCTLIST);

            if(outputProductList[0].get('material') != null) ⑤
            {
                String restrictedProdMaterial = (String) outputProductList[0].get('material');

                if(!String.isEmpty(restrictedProdMaterial))
```

```
                {
                    ⑤
                    Map<String,Object> accountData = ccrz.ccApiAccount.fetch(new
Map<String,Object>{
                        ccrz.ccApi.API_VERSION => inputData.get(ccrz.ccApi.API_VERSION),
                        ccrz.ccApiAccount.ID => ccrz.cc_CallContext.effAccountId,
                        ccrz.ccAPI.SIZING => new Map<String, Object>{
                            ccrz.ccAPIProduct.ENTITYNAME => new Map<String, Object>{
                                    ccrz.ccAPI.SZ_DATA => ccrz.ccAPI.SZ_XL
                            }
                        }
                    });

                    List<Map<String, Object>> outputAccountList = (List<Map<String, Object>>)
accountData.get(ccrz.ccAPIAccount.ACCOUNTS); ⑤

                    if(outputAccountList[0].get('restrictedMaterial') != null) ⑥
                    {
                        String restrictedMaterialAccount = (String)
outputAccountList[0].get('restrictedMaterial'); ⑥

                        if(!String.isEmpty(restrictedMaterialAccount))
                        {
                            if(restrictedMaterialAccount.equals(restrictedProdMaterial)) ⑦
                            {
                                ccrz.cclog.log('restricted product for account'); ⑦

                                restrictedProduct = true;
                            }
                        }
                    }
                }
            }
        }

        // empty out the line item data if the product is restricted
        if (restrictedProduct)
        {
            ⑦
            inputData.put(ccrz.ccAPICart.LINE_DATA,null);
        }

        return super.processInputData(inputData);
    }

}
```

i. **Configure CloudCraze to use your new Logic Provider**

1. In your Salesforce Org, click **CC Admin** from the main tab.
2. On the right side dropdown select the **DefaultStore** Storefront.
3. On the left side bar, select `Service Management`
4. Scroll down to the Logic provider section and search for ccLogicCartAddTo
5. Set the class name to `c.ccTrainingLogicCartAddTo`
6. Save your changes

j. **Verify your work in the storefront**

1. Search for the product you edited in a prior step in your storefront - this would be the product the user is restricted from adding to the cart.
2. Try to add this product to the cart on either the Product Detail Page or the Product Listing Page.
3. You should observe the product was not added to the cart.
4. Try to add another product with no restricted material referenced on it and it should be successfully added to the cart.

3.4.2. Exercise: Override the Cart Validate Logic Provider

The `ccLogicCartValidate` class allows for specific business validation, messaging, and processing on the cart page.

In this exercise, we'll show you how to inject custom logic into the life cycle of the Cart details page.

Use Case

1. DefaultStore would like to prevent users from placing orders that would cause them to exceed the Credit Limit associated with their Account.
2. When this occurs, the User should be notified as to the reason they are not able to proceed to checkout.

Prior to this exercise, you must create a Custom Field of type Currency with an API name of `CreditLimit__c` on the Account object. You can ignore this step if this field has been created in a prior exercise.

a. **Add a custom field to the Account object**

   1. Create a new attribute for credit limit on the Account object.
   2. While in the setup menu, Type `Object` in the search box, and then select the `Objects` option under the `Create` menu.
   3. Scroll down to the `Custom Fields and Relationships` section and then select `New`
   4. Select `Currency` as the data type on the next screen
   5. Set the following on the subsequent screen.
      i. Field Label: Credit Limit
      ii. Length: 16
      iii. Field Name: CreditLimit
      iv. Decimal Places : 2
   6. Make the field visible for all the profiles.
   7. Add the `Credit Limit` attribute to the layout for the `Account` object and save your changes.

b. **Implement your new Cart Validate Logic Override**

   1. Create a new Apex class called `ccTrainingLogicCartValidate` that extends `ccrz.ccLogicCartValidate`
   2. Override the `processValidate` method.
   3. Create a private method named `checkAccountCreditLimit` that takes one argument of type `Map<String,Object>`.
   4. Inside the `checkAccountCreditLimit` method, check if the total amount on the cart is less than or equal to the credit limit for the current Account

5. If the total amount is greater than the credit limit, the `ALLOW_CHECKOUT` parameter should be set to false. And if the total amount is not greater than the credit limit, `ALLOW_CHECKOUT` parameter should be set to true.
6. You should also create a page message that communicates why the checkout button is hidden from the user.

```
global with sharing class ccTrainingLogicCartValidate extends ccrz.ccLogicCartValidate{ ①

  private Boolean checkAccountCreditLimit(Map<String, Object> inputData) { ③
    Decimal accountCreditLimit = ccrz.cc_CallContext.currAccount.CreditLimit__c;
    List<Object> cartObjList = (List<Object>) inputData.get(ccrz.ccApiCart.CART_OBJLIST);
    ccrz__E_Cart__c cartRecord = (ccrz__E_Cart__c) cartObjList.get(0);
    if (cartRecord.ccrz__SubTotalAmount__c <= accountCreditLimit) { ④
      return true;
    } else {
      return false;
    }
  }

  global override Map<String, Object> processValidate(Map<String, Object> inputData) { ②
    Map<String, Object> cartFetchRequest = new Map<String, Object>(inputData);
    cartFetchRequest.put(ccrz.ccApi.SIZING,
        new Map<String, Object> {
            ccrz.ccApiCart.ENTITYNAME => new Map<String, Object> {
                ccrz.ccApi.SZ_DATA => ccrz.ccApi.SZ_L // LARGE is the smallest size OOTB that will
return for us SubtotalAmount__c
                ,ccrz.ccApi.SZ_ASSC => FALSE // for this override we are only interested in total
amount and not any details of individual line items
                ,ccrz.ccApi.SZ_SKIPTRZ => TRUE // skip transform so, we can work directly with the
sObject record here in the backend.
            }
        }
    );

    Map<String, Object> cartFetchResponse = ccrz.ccApiCart.fetch(cartFetchRequest);
    Boolean shouldAllowCheckout = checkAccountCreditLimit(cartFetchResponse);

    List<ccrz.cc_bean_Message> retMessages = new List<ccrz.cc_bean_Message>();

    if(!shouldAllowCheckout) {
      ccrz.cc_bean_Message msg = new ccrz.cc_bean_Message(); ⑥
      msg.labelId              = 'CCTraining_CreditExceeded';
      msg.type                 = ccrz.cc_bean_Message.MessageType.CUSTOM;
      msg.severity             = ccrz.cc_bean_Message.MessageSeverity.ERROR;
      msg.classToAppend        = 'messagingSection-Info';

      retMessages.add(msg);
    }

    inputData.put(ccrz.ccApiCart.ALLOW_CHECKOUT, shouldAllowCheckout); ⑤
    inputData.put(ccrz.ccApi.MESSAGES, retMessages);

    return inputData;
  }
}
```

c. **Configure CloudCraze to use your new ccLogicCartValidate override**

1. In your Salesforce Org, click **CC Admin** from the main tab.
2. On the right side dropdown select the **DefaultStore** Storefront.
3. On the left side bar, near the middle, under "Integrations" section, click on **Service Management**.
4. Set the `ccLogicCartValidate` to `c.ccTrainingLogicCartValidate`

d. **Create a page label with the following values**

   i. Page Name : Cart

   ii. Page Label Name : CCTraining_CreditExceeded

   iii. Value : You have exceeded your available credit. Please request a credit override or reduce the amount on your order

   iv. Storefront : **DefaultStore**

e. **Rebuild your page label cache**

f. **Verify your changes**

   i. Add a product to the cart that exceeds your credit limit

   ii. Observe that the Checkout button will no longer be visible

   iii. You'll also notice that Checkout button will be visible if the cart total doesn't exceed the credit limit.

`ccrz.ccApiCart.validate` (and in turn `ccLogicCartValidate`) will only be called from the OOTB storefront if the `c.skext` (Cart: Skip Cart Validation) Configuration Setting is set to `FALSE`.

3.5. REST APIs

CloudCraze Global APIs are wrapped and surfaced for REST by the CloudCraze REST API Wrappers. These classes build on the Force.com APEX Rest class approach in the following manner.

Note: CloudCraze recommends only using the REST APIs via an integration user. Because of the need to enable API access on the user profile/permission set CloudCraze does not recommend allowing access to the REST APIs for external users.

Routing: Request Path Parsing

The REST wrappers create method endpoints through the annotation system shown below. `e.g. @RestResource(urlMapping='/cccart/x')` This essentially gnerates a value of /cccart/x when the RestContext.request.resourcePath is used.

Versioning

Versioning is handled through the requestURI routing system mentioned above. The version will be prefixed with a `v` and the `route/integer` will follow.

Minimum Version

The minimum version of the API that can be invoked is currently version 1. In the future, the minimum version may be increased. Invoking a REST API with a version lower than that (e.g. `0`) will return a HTTP 400 response with messaging indicating that the version below the minimum version was requested.

Maximum Version

The maximum version of the API that can be invoked is currently version 4. In the future, the maximum version may be increased. Invoking a REST API with a version higher than the maximum (e.g. `9001`) will return a HTTP 400 response with messaging indicating that the version above the maximum was requested.

Unavailable Version

Other than the general restriction imposed on API's via the minimum and maximum avaialable versions. There are some API's that begin at a particular version E.g. Foo API only honors requests starting at version 3.In cases where the API is invoked with a lower version number, a HTTP 500 response will be returned and it will contain a message noting that the requsted version is unavailable.

3.6. Global Extension Points

CloudCraze provides several extension points within the core product where partners can extend native functionality.

Listed below are some of the Global Global Extension Points

- ccrz.cc_api_CartExtension
- ccrz.cc_api_DeliveryDate
- ccrz.cc_api_InventoryExtension
- ccrz.cc_api_OutboundOrderCancel

- ccrz.cc_api_PriceAdjustment
- ccrz.cc_api_ProductQuantityRule
- ccrz.cc_api_ShippingAndHandling
- ccrz.cc_hk_Catalog
- ccrz.cc_hk_Category
- ccrz.cc_hk_DynamicTheme
- ccrz.cc_hk_EffectiveAccount
- ccrz.cc_hk_Invoice
- ccrz.cc_hk_Menu
- ccrz.cc_hk_Order
- ccrz.cc_hk_Payment
- ccrz.cc_hk_Pricing
- ccrz.cc_hk_Promotion
- ccrz.cc_hk_SSO
- ccrz.cc_hk_Subscriptions
- ccrz.cc_hk_TaxCalculation
- ccrz.cc_hk_UserInterface
- ccrz.cc_if_OutboundOrder
- ccrz.cc_hk_DynamicTheme – setting a theme dynamically
- ccrz.cc_hk_UserInterface
- ccrz.cc_hk_Menu – fetches all the menu items from the public cache
- ccrz.cc_hk_Catalog – filters out specific categories
- ccrz.cc_hk_Category – returns the category tree or just sub categories
- ccrz.cc_hk_Promotion – filter out promotions
- ccrz.cc_api_ProductQuantityRule – retrieves rules that can be applied to the cart
- ccrz.cc_api_InventoryExtension – returns the inventory
- ccrz.cc_hk_Pricing – returns pricing

3.6.1. Extension Implementation

Implementing an extension is a two step process:

1. First, we need to create a new class that extends the base implementation in CloudCraze
    i. Example: `global class ccTraining_hk_Menu extends ccrz.cc_hk_Menu { }`
2. Configure the extension in CC Admin to be used on CloudCraze which is on a per-storefront basis.
    i. Remember to prefix the class name with `c.` as this lets CloudCraze know it's not part of the managed

package.

3.7. Extending My Account Page

Use Case

1. Create an Apex controller that will be referenced in your page include
2. Create a new Page Include for the My Account Page and reference the controller above on it.
3. Update the `getFieldsMap` method for the data service provider for Accounts
4. Configure CloudCraze to use your new Page Include
5. Verify your changes

Prior to this exercise, ensure you have a custom field of type Currency on the account object with an API name of `CreditLimit__c`

a. **Implement your new Apex Controller**

1. Create a new Apex class called `ccTrainingMyAccountCtrl`
2. Define two variables named `remainingCredit` and `userCurrency`
3. Define a constructor for this class
4. Fetch the account detail of the user using `ccrz.ccAPIAccount.fetch`
5. Set the value from the `CreditLimit__c` field to the `remainingCredit` variable
6. Set the value from `ccrz.cc_CallContext.userCurrency` on the `userCurrency` variable

```
global with sharing class ccTrainingMyAccountCtrl ①
{
    global Decimal remainingCredit{get;private set;} ②

    global String userCurrency{get;private set;} ②

    global ccTrainingMyAccountCtrl() ③
    {
        ④
        List<Object> acctRes = (List<Object>)ccrz.ccApiAccount.fetch(new Map<String,Object>{
       ccrz.ccApi.API_VERSION => ccrz.ccApi.CURRENT_VERSION,
             ccrz.ccApiAccount.ID => ccrz.cc_CallContext.currAccountId,
       ccrz.ccApi.SIZING=>new Map<String, Object>{
         ccrz.ccApiAccount.ENTITYNAME => new Map<String, Object>{
           ccrz.ccAPI.SZ_SKIPTRZ=>true
         }
       }
    }).get(ccrz.ccApiAccount.ACCOUNTS);

    Account account = null;

        if(acctRes.get(0) != null)
        {
            account = (Account)acctRes.get(0);

            remainingCredit = account.CreditLimit__c; ⑤
        }

        userCurrency = ccrz.cc_CallContext.userCurrency; ⑥
    }
}
```

b. **Implement your new Page Include**

1. Create a new VisualForce page named `ccTrainingMyAccountBIE`

    i. In your SalesForce Org, click the **Developer Console** link under the dropdown menu **under your name** - this should be on the top right hand corner on the page.

    ii. A new window should pop-up.

    iii. In the Developer Console menu, select: **File → New → VisualForce Page**

    iv. In the popup window (New Apex Page), enter **"ccTrainingMyAccountBIE"** as your page name and click **ok.**

    v. Remove the boilerplate code (delete the contents of the page).

    vi. Enter the following: please note that the code below was added for your convenience. It is also available on the wiki here

```
<apex:page docType="html-5.0" sidebar="false" showHeader="false" standardStylesheets="false"
applyHtmlTag="false" controller="ccTrainingMyAccountCtrl">

    <script>
        CCRZ.uiProperties.contactInfoView.desktop.tmpl = 'CCTrainingMyAccount-
ContactInformation-Desktop'; ①
    </script>
```

```
        ②
   <script id="CCTrainingMyAccount-ContactInformation-Desktop" type="text/template">
      <div class="panel panel-default cc_panel cc_myaccount_profile">
        <div class="panel-body cc_body cc_myaccount_content">
          <h3 class="cc_title">{{pageLabelMap 'MyAccount_Profile'}}</h3>
          {{#ifEquals this.commerceType "B2B"}}
            <div class="panel panel-default cc_panel cc_myaccount_information">
              <div class="panel-heading cc_heading">
                <h3 class="panel-title cc_title">{{pageLabelMap
'MyAccount_Profile_Account_Information'}}</h3>
              </div>
              <div class="panel-body cc_body cc_myaccount_general">
                <p class="myAccProfileNote cc_profile_note">
                  {{pageLabelMap 'MyAccount_Profile_Note'}}
                </p>
                {{#ifDisplay 'reg.addlInf'}}
                  <p class="myAccProfileCompany cc_profile_company">
                    <span class="cc_profile_company_label">{{pageLabelMap
'MyAccount_Profile_Company'}}&#58;</span>
                    <span class="cc_profile_company_valuel">{{accountBean.name}}</span>
                  </p>
                {{/ifDisplay}}
                <!-- Exercise TODO - Add your code to this section -->
                                <p class="myAccProfileCompany cc_profile_company">
                    <span class="cc_profile_company_label">{{pageLabelMap
'CCTrainingMyAccount_Remaining_Credit'}}&#58;</span>
                    <span class="cc_profile_company_valuel"></span> ③
                </p>
                <p class="myAccAccountGroup cc_profile_account_group">
                  <span class="cc_profile_account_group_label">{{pageLabelMap
'MyAccount_Profile_Account_Group'}}&#58;</span>
                  <span
class="cc_profile_account_group_value">{{accountBean.accountGroupName}}</span>
                </p>
                <p class="myAccProfilePhone cc_profile_phone">
                  <span class="cc_profile_phone_label">{{pageLabelMap
'MyAccount_Profile_Phone'}}&#58;</span>
                  <span class="cc_profile_phone_value">{{accountBean.phone}}</span>
                </p>
                <div class="row">
                  <div class="col-md-6 myAccBillingAddr cc_billing_address">
                    <span class="cc_profile_billing_label">{{pageLabelMap
'MyAccount_Profile_Account_Billing_Address'}}</span>
                    <span class="cc_profile_billing_value">{{> addressDisplay
this.accountBean.billingAddress}}</span>
                  </div>
                  <div class="col-md-6 myAccShippingAddr cc_shipping_address">
                    <span class="cc_profile_shipping_label">{{pageLabelMap
'MyAccount_Profile_Account_Shipping_Address'}}</span>
                    <span class="cc_profile_shipping_value">{{> addressDisplay
this.accountBean.shippingAddress}}</span>
                  </div>
                </div>
              </div>
            </div>
            <div class="panel panel-default cc_panel cc_myaccount_contact_information">
              <div class="panel-heading cc_heading">
                <h3 class="panel-title cc_title">{{pageLabelMap
'MyAccount_Profile_Contact_Information'}}</h3>
```

```
            </div>
            <div class="panel-body cc_body cc_myaccount_contact">
              <p class="myAccProfileName cc_profile_name">
                <span class="cc_profile_name_label">{{pageLabelMap
'MyAccount_Profile_Name'}}&#58;</span>
                {{#if contactBean.firstName}}
                  <span class="cc_profile_name_value">{{contactBean.firstName}}
{{contactBean.lastName}}</span>
                {{else}}
                  <span class="cc_profile_name_label">No name stored.</span>
                {{/if}}
              </p>
              <p class="myAccProfilePhone cc_profile_phone">
                <span class="cc_profile_phone_label">{{pageLabelMap
'MyAccount_Profile_Phone'}}&#58;</span>
                {{#if contactBean.phone}}
                  <span class="cc_profile_phone_value">{{contactBean.phone}}</span>
                {{else}}
                  <span class="cc_profile_phone_value">No phone number stored.</span>
                {{/if}}
              </p>
              <div class="row">
                <div class="col-md-6 myAccMailingAddr cc_mailing_address">
                  <span class="cc_profile_mailing_label">{{pageLabelMap
'MyAccount_Profile_Contact_Mailing_Address'}}</span>
                  <span class="cc_profile_mailing_value">{{> addressDisplay
this.contactBean.mailingAddress}}</span>
                </div>
                <div class="col-md-6  myAccOtherAddr cc_other_address">
                  <span class="cc_profile_other_label">{{pageLabelMap
'MyAccount_Profile_Contact_Other_Address'}}</span>
                  <span class="cc_profile_other_valuel">{{> addressDisplay
this.contactBean.otherAddress}}</span>
                </div>
              </div>
            </div>
          </div>
        {{/ifEquals}}
        <div class="panel panel-default cc_panel cc_myaccount_user_information">
          <div class="panel-heading cc_heading">
            <h3 class="panel-title cc_title">{{pageLabelMap
'MyAccount_Profile_User_Information'}}</h3>
          </div>
          <div class="panel-body cc_body cc_myaccount_user">
            <p class="myAccName cc_name">
              <span class="cc_profile_name_label">{{pageLabelMap
'MyAccount_Profile_Name'}}&#58;</span>
              <span class="cc_profile_name_value">{{userFirstName}} {{userLastName}}</span>
            </p>
            <p class="myAccPhone cc_acct_phone">
              <span class="cc_acct_phone_label">{{pageLabelMap
'MyAccount_Profile_Phone'}}&#58;</span>
              <span class="cc_acct_phone_value">{{userPhone}}</span>
            </p>
            <p class="myAccUserName cc_acct_username">
              <span class="cc_acct_username_label">{{pageLabelMap
'MyAccount_Profile_Username'}}&#58;</span>
              <span class="cc_acct_username_value">{{username}}</span>
            </p>
```

```
              <p class="myAccEmailAddr cc_acct_email">
                <span class="cc_acct_email_label">{{pageLabelMap
'MyAccount_Profile_Email'}}&#58;</span>
                <span class="cc_acct_email_value">{{emailAddress}}</span>
              </p>
              <p class="myAccLanguage cc_acct_language">
                <span class="cc_acct_language_label">{{pageLabelMap
'MyAccount_Profile_Language'}}&#58;</span>
                <span class="cc_acct_language_value">{{language}}</span>
              </p>
              <p class="myAccCurrency cc_acct_currency">
                <span class="cc_acct_currency_label">{{pageLabelMap
'MyAccount_Profile_Currency'}}&#58;</span>
                <span class="cc_acct_currency_value">{{currencyName}}</span>
              </p>
              {{#ifDisplay 'reg.tmZn'}}
                <p class="myAccTimeZone cc_acct_timezone">
                  <span class="cc_acct_timezone_label">{{pageLabelMap
'MyAccount_Profile_TimeZone'}}&#58;</span>
                  <span class="cc_acct_timezone_value">{{timeZone}}</span>
                </p>
              {{/ifDisplay}}
            </div>
          </div>
          {{#if hideEditProfile}}
          {{else}}
            <input type="button" class="btn btn-default btn-sm gotoSectionContactInfoEdit
cc_edit_profile" value="{{pageLabelMap 'MyAccount_EditProfile'}}" />
          {{/if}}
        </div>
    </div>
 </script>


</apex:page>
```

① Handlebars template override, this template override can also be done in the uiproperties.js file

② Handlebars template ID defined here should always match the value in the override from the previous step.

③ Placeholder for the code that should be added as part of this exercise. See below for more details.

c. **Expose the value of the remaining credit limit on this page**

   i. Copy and paste the below to the annnotated section in the VisualForce page described above. The code should essentially go in between the span tag.

   ii. `{{price '{!remainingCredit}' '{!userCurrency}'}}`

d. **Create a page label with the following values**

Page Name

All

Page Label Name

CCTrainingMyAccount_Remaining_Credit

Value

Remaining Credit

Storefront

DefaultStore

e. **Rebuild your page label cache**

f. **Update the data service provider for Accounts**

1. Search for `ccTrainingServiceAccount` in your developer console or IDE **(This should already be in place, if not please create this class and ensure it extends ccrz.ccServiceAccount)**
2. Override the `getFieldsMap` method **(This should already be in place, if not please complete this step).**
3. Inside the `getFieldsMap` override, retreive the base fields this class returns by default **(This should already be in place).**
4. While still inside this method, concatenate `CreditLimit__c` to the String that contains the default base fields
5. Return the newly created String.
6. Save your changes

```
global with sharing class ccTrainingServiceAccount extends ccrz.ccServiceAccount ①
{
    global virtual override Map<String, Object> getFieldsMap(Map<String, Object> inputData) ②
    {
        inputData = super.getFieldsMap(inputData); ③

        String objectFields = (String)inputData.get(ccrz.ccService.OBJECTFIELDS); ③

        objectFields += ',CreditLimit__c,RestrictedMaterial__c' ; ④

        return new Map <String,Object> {ccrz.ccService.OBJECTFIELDS => objectFields}; ⑤
    }
}
```

g. **Configure CloudCraze to use your new Page Include**

1. In your Salesforce Org, click **CC Admin** from the main tab.
2. On the right side dropdown select the **DefaultStore** Storefront.
3. On the left side bar, near the top, under **Settings** section, click on **Configuration Settings**.
4. Click on **New** button to create a new Configuration Setting and enter the following to **enable** the page include:

When filling out the value for the `Page` input field referenced below, you should keep the following in mind. Start out by typing the name of the page of interest. E.g Type `My Account` and then select the appropriate value from the autocompleted suggestions in the input field.

Module

Body Includes End

Configuration

Enabled

Page

My Account (MA)

Value

TRUE

5. Select **New** to create a new Configuration Setting and enter the following to **set** the page include:

Module

Body Includes End

Configuration

Page Include Name

Page

My Account (MA)

Value

c__ccTrainingMyAccountBIE

6. Create and Activate a new Configuration Cache:
    i. On the right side dropdown, click on the **Global Settings** link.
    ii. On the left side bar, select **Configuration Cache Management**.
    iii. Click on **Build New**. After a few seconds, click on **Refresh List**. A new configuration cache should have been built. Click on **Activate** to enable it.

h. **Verify your changes on the My Account Page**
    i. Add any value to the credit limit field on your account record
    ii. Verify that the amount is displayed correctly on **My Account → Contact Information**

i. **Verify your changes on the Cart page**
    i. Add a product to the cart that exceeds your credit limit
    ii. Observe that the Checkout button will no longer be visible
    iii. You'll also notice that Checkout button will be visible if the cart total doesn't exceed the credit limit.

3.8. Extending the Cart Page

Use Case

1. Create new attributes on the Case and CC Cart to track the requested Credit Override.
2. Create an Apex controller that will be referenced on your page include.
3. Create a Page Include for the Cart Page.
4. Configure CloudCraze to use your new Page Include.
5. Verify your changes

a. **Add a custom field to the Case object**

1. Create a new attribute for linking a case to its source cart on the Case object.
2. While in the setup menu, Type `Case` in the quick find box.
3. Select **Fields** under the **Build → Customize → Cases → Fields** menu.
4. Scroll down to the `Custom Fields and Relationships` section and then select `New`
5. Select `Lookup Relationship` as the Data Type on the next screen
6. Select `CC Cart` as the **Related to** field on the next screen
7. Set the following on the subsequent screen.
    i. Field Label: Reference Cart
    ii. Field Name: SourceCart
    iii. Leave other fields as is in their default state
8. Make the field visible for all the profiles.
9. Keep advancing through the setup screens and then save your changes.

b. **Add a custom field to the Cart object**

1. Create a new attribute for indicating the state of a credit override on the Cart Object
2. While in the setup menu, Type `Object` in the quick find box.
3. Select **Objects** under the **Build → Create → Objects** menu.
4. Select **CC Cart** from the list of the available custom objects. **Do not** select the Edit button.
5. Scroll down to the `Custom Fields and Relationships` section and then select `New`
6. Select `Checkbox` as the Data type on the next screen
7. Set the following on the subsequent screen.
    i. Field Label: Credit Override Requested
    ii. Field Name: CreditOverrideRequested
    iii. Default Value: Unchecked

iv. Leave other fields as is in their default state

8. Make the field visible for all the profiles.
9. Keep advancing through the setup screens and then save your changes.

c. **Implement your new Apex Controller**

1. Create a new Apex class called `ccTrainingCartCtrl`
2. Create a dummy merge variable named `blank` with a getter and a setter
3. Create a remote action method named `createCreditCase` that accepts one argument of type `ccrz.cc_RemoteActionContext`
4. Inside the `createCreditCase` method, do the following.
   i. Initialize the remote action context
   ii. Create an instance variable of type `cc_RemoteActionResult`
   iii. Set the default values for the variable referenced in the above step.
5. Create a List of type `String` and add the encrypted ID of the cart to this list
6. Invoke `ccrz.ccAPICart.fetch` by passing in the appropriate input data of type `Map<String,Object>`
7. Extract the list of cart objects from the response of the fetch call.
8. Verify that the API returned at least one record
9. Set the value on the `CreditOverrideRequested__c` field to TRUE
10. Create a `case` and set its values to the following. They are currently set to their default values in the provided snippet. You'll want them to replace the default values with the below.

    Origin

    'Web';

    Reason

    'Credit Override'

    Type

    'Cart'

    Subject

    'Credit override request for '+ ccrz.cc_CallContext.effAccount.Name

    SourceCart__c

    This value was left out intentionally,the reference to the cart needs to be set here

    ContactId

    ccrz.cc_CallContext.currUser.ContactId

    AccountId

    ccrz.cc_CallContext.effAccountId

11. Insert the `case` record in Salesforce
12. Update the cart record in Salesforce
13. Set the value of the success variable of type `ccrz.cc_RemoteActionResult` to true.

```
global with sharing class ccTrainingCartCtrl ①
{
   global static String blank {get{return ''; } private set;} ②

    @RemoteAction
    global static ccrz.cc_RemoteActionResult createCreditCase(final ccrz.cc_RemoteActionContext
ctx) ③
    {
        ccrz.cc_RemoteActionResult res = ccrz.cc_CallContext.init(ctx); ④
        res.success = false; ④
        res.inputContext = ctx; // < ④

        try
        {
            List<String> cartidlist = new List<String>{};
            cartidlist.add(ctx.currentCartId); ⑤

            ⑥
            Map<String, Object> cartApiFetchResponse = ccrz.ccApiCart.fetch(new
Map<String,Object>{
       ccrz.ccApi.API_VERSION => ccrz.ccApi.CURRENT_VERSION,
             ccrz.ccApiCart.CART_ENCIDLIST => cartidlist,
       ccrz.ccApi.SIZING=>new Map<String, Object>{
         ccrz.ccApiCart.ENTITYNAME => new Map<String, Object>{
           ccrz.ccAPI.SZ_SKIPTRZ=>true
        } }
      });

            ⑦
            List<ccrz__E_Cart__c> cartRes =
(List<ccrz__E_Cart__c>)cartApiFetchResponse.get(ccrz.ccApiCart.CART_OBJLIST);

            ccrz__E_Cart__c theCart = null;

            if(cartRes.get(0) != null) ⑧
            {
                theCart = (ccrz__E_Cart__c)cartRes.get(0);
                ccrz.ccLog.log(LoggingLevel.DEBUG,
'ccTrainingCartCtrl:createCreditCase:theCart', theCart);
                theCart.CreditOverrideRequested__c = TRUE; ⑨

                Case caseRequest = new Case();  ⑩
                // Exercise TODO
                caseRequest.Origin         =  '' ; ⑩
                caseRequest.Reason         = '' ; ⑩
                caseRequest.Type           = '' ; ⑩
                caseRequest.Subject        =  '' ; ⑩
                caseRequest.SourceCart__c  =  null; ⑩
                caseRequest.ContactId      = null; ⑩
                caseRequest.AccountId      =  null; ⑩

                insert caseRequest; ⑪
```

```
                update theCart; ⑫

                res.success = true; ⑬
            }
        }
        catch(Exception e)
        {
            res.data = e;
            res.success = false;
        } finally {
            ccrz.ccLog.close(res);
        }

        return res;
    }
}
```

d. **Implement your new Page Include**

A defect was intentionally introduced in the reference code provided for the Cart page. You need to resolve this issue in order to verify your changes as described below.

1. Create a new VisualForce page named `ccTrainingCartBIE`
   i. In the Developer Console menu, select: **File → New → VisualForce Page**
   ii. In the popup window, select (New Apex Page), then enter **ccTrainingCartBIE** as your page name. Save your changes by selecting **ok**.
   iii. Remove the boilerplate code (delete the contents of the page).
   iv. Add the code snippet below to your new page : please note that the code below was added for your convenience. It is also available on the wiki here
   v. Conversely, you can also access the code by copying over the source content via your browser menu
      A. In your Chrome browser, you can navigate to **View → Developer → View Source**
   vi. The controller `ccTrainingCartCtrl` created in the previous step is already linked to this page.

```
<apex:page docType="html-5.0" sidebar="false" showHeader="false" standardStylesheets="false"
applyHtmlTag="false" controller="ccTrainingCartCtrl">  ①
   {!blank}
  <script type="text/javascript">
        CCRZ.uiProperties.CartDetailView.partials.actionsTotals = '#CCTrainingCartDetail-
ActionTotals-View'; ②
 </script>

<script id="CCTrainingCartDetail-ActionTotals-View" type="text/template"> ③
   <div class="row">
      <div class="col-md-5">
         {{#ifStoreSetting 'Display_Cart_Coupon__c'}}
            <div class="panel panel-default cc_panel cc_shopping_cart_discount_panel">
               <div class="panel-heading cc_heading">
                  <h3 class="panel-title cc_title">{{pageLabelMap 'CartInc_discountCodes'}}</h3>
               </div>
               <div class="cc_myaccount_content panel-body cc_body">
```

```
                {{#if this.hasCoupon}}
                   <form id="couponClearForm" class="cc_coupon_clear_form">
                      <div class="discount_code cc_discount_code">
                         <p class="cc_applied_coupon">{{pageLabelMap 'CartInc_AppliedCoupon'}}
{{this.couponName}}</p>
                         <p class="cc_clear_coupon_code"><button id="clearCouponBtn"
type="button" class="btn btn-default btn-sm cc_clear_coupon">{{pageLabelMap
'CartInc_ClearCoupon'}}</button></p>
                      </div>
                   </form>
                {{else}}
                   <form id="couponAddForm" class="cc_coupon_add_form">
                      <div class="discount_code cc_discount_code">
                         <p id="couponAddError" class="cc_coupon_add_error"></p>
                         <p class="cc_enter_coupon_code">{{pageLabelMap
'CartInc_EntercouponCode'}}</p>
                         <div class="form-group">
                            <label for="addCouponId" class="sr-only">Add Coupon</label>
                            <input type="text" id="addCouponId" name="couponId" class="form-
control cc_add_coupon_id" />
                            <div class="couponMessagingSection-Error"></div>
                         </div>
                         <button id="addCouponBtn" type="button" class="btn btn-default btn-sm
cc_add_coupon">{{pageLabelMap 'CartInc_ApplyCoupon'}}</button>
                      </div>
                   </form>
                {{/if}}
             </div>
          </div>
       {{/ifStoreSetting}}
    </div>
    <div class="col-md-7">
       <div class="row">
          <div class="col-md-12">
             <div class=" cc_action_totals pull-right">
                <button class="btn btn-default btn-sm continueShoppingButton
cc_continue_shopping_button" name="" type="button" >{{pageLabelMap
'CartInc_ContinueShopping'}}</button>
                {{#if {!ISPICKVAL($User.UserType, 'Guest')} }}
                {{ else }}
                   {{#ifStoreSetting 'AutoCalcPriceAdjust__c'}}
                   {{else}}
                      <button class="btn btn-default btn-sm getExtPricingButton
cc_get_ext_pricing_button" type="button" >{{pageLabelMap 'CartInc_ApplyDiscounts'}}</button>
                   {{/ifStoreSetting}}
                   {{#ifDisplay 'WL.PkrOn'}}
                   <button class="btn btn-default btn-sm cc_create_cart_wishlist" data-
toggle="modal" data-target="#wishMod" name="" type="button" >{{pageLabelMap
'Create_Cart_Wishlist'}}</button>
                   {{/ifDisplay}}
                {{/if}}
                <button class="btn btn-default btn-sm updateCartButton cc_update_cart_button"
name="" type="button" >{{pageLabelMap 'CartInc_Update'}}</button>
                {{#if {!ISPICKVAL($User.UserType, 'Guest')} }}
                   {{#ifDisplay 'C.EmailAnon'}}
                      <a href="#emailModal" id="emailCartLink" class="cc_email_cart_link" data-
toggle="modal"><button class="btn btn-default btn-sm remove cc_email_cart_link_button"
type="button" >{{pageLabelMap 'CartOrderEmailer_Header'}}</button></a>
                   {{/ifDisplay}}
```

```
                {{else}}
                   {{#ifDisplay 'C.EnableEmailCart'}}
                      <a href="#emailModal" id="emailCartLink" class="cc_email_cart_link" data-
toggle="modal"><button class="btn btn-default btn-sm remove cc_email_cart_link_button"
type="button" >{{pageLabelMap 'CartOrderEmailer_Header'}}</button></a>
                   {{/ifDisplay}}
                {{/if}}
                <!-- rfq button -->
                {{#if {!ISPICKVAL($User.UserType, 'Guest')} }}
                   {{#ifDisplay 'rfq.GuestUser'}}
                      <a href="#rfqModal" id="rfqLink" class="cc_rfq_link" data-
toggle="modal"><button class="btn btn-default btn-sm remove cc_rfq_link_button" type="button"
>{{pageLabelMap 'CartRFQForm_Header'}}</button></a>
                   {{/ifDisplay}}
                {{else}}
                   {{#ifDisplay 'rfq.LoggedIn'}}
                      <a href="#rfqModal" id="rfqLink" class="cc_rfq_link" data-
toggle="modal"><button class="btn btn-default btn-sm remove cc_rfq_link_button" type="button"
>{{pageLabelMap 'CartRFQForm_Header'}}</button></a>
                   {{/ifDisplay}}
                {{/if}}
                <!-- rfq button -->
                {{#ifNotEquals this.allowCheckout true}} //  ④
                     <!-- Exercise TODO  - Add the HTML code for the credit override button here
-->
                  {{/ifNotEquals}}
             </div>
          </div>
       </div>
       <div class="row">
          <div class="col-md-12">
             <ul class="checkout list-unstyled cc_checkout pull-right">
                {{#if subTotal}}
                   <li class="grand_total cc_grand_total">
                      <p class="price cc_price">
                         <span class="cc_label">{{pageLabelMap 'CartInc_Subtotal'}}:</span>
                         <span class="cc_value">{{{price subTotal}}}</span>
                      </p>
                   </li>
                {{/if}}
                {{#if this.totalInfo}}
                   <li class="cart_total_amount cc_cart_total_amount">{{this.totalInfo}}</li>
                {{/if}}
                <li class="totalsmessagingSection cc_total_messaging_section"></li>
                {{#if this.cartItems}}
                   {{#ifNotEquals this.cartItems.length 0}}
                      {{#if this.allowCheckout}}
                         {{#ifEquals this.preventCheckout false}}
                            <li>
                               <button type="button" class="btn btn-default btn-sm checkOutBtn
cc_checkout_btn pull-right">{{pageLabelMap 'CartInc_Checkout'}}</button>
                            </li>
                         {{/ifEquals}}
                      {{/if}}
                   {{/ifNotEquals}}
                {{/if}}
             </ul>
          </div>
       </div>
```

```
      </div>
   </div>
</script>

<script type="text/javascript">
  jQuery(function($)
    {
        CCRZ.subsc = _.extend(CCRZ.subsc||{}); ⑤

        ⑥
          CCRZ.subsc.cartRemoteActions = _.extend({
            className : 'ccTrainingCartCtrl',
            createCreditCaseAction : function(callback)
            {
                this.invokeCtx('createCreditCase',
                    function(resp)
                    {
                        callback(resp);
                    },
                    {
                        buffer : false, // this call will be executed by itself
                        nmsp   : false  // defines that this is a call to a subscriber class
                    }); // end invokeCtx call
             }
           },CCRZ.RemoteInvocation);
           ⑥

        CCRZ.pubSub.on('view:CartDetailView:refresh', function(cartDetailView) ⑦
        {
          cartDetailView.createCreditCaseAction = function(event)  ⑧
           {
               CCRZ.subsc.cartRemoteActions.createCreditCaseAction(function(resp)
                {
                  if(resp && resp.success)
                  {
                      ⑨
                     CCRZ.pubSub.trigger('pageMessage',{
                                   messages:[
                                       {
                                           type : 'CUSTOM',
                                           severity : 'INFO',
                                           classToAppend : 'messagingSection-Info',
                                           labelId : 'CCTraining_CreditOverrideRequested'
                                       }
                                   ]
                        });
                   }
                });
            };

           cartDetailView.events['click .createCreditCase'] = 'createCreditCaseAction'; ⑩
           cartDetailView.delegateEvents(); ⑪
        }};
    });
</script>
</apex:page>
```

① The Apex controller leveraged by this page is referenced here

② Handlebars template override, this template override can also be done in the uiproperties.js file.

③ Handlebars template ID defined here should always match the value in the override from the previous step. The next step is to Add the new credit override button to this page.

④ Placeholder for where the credit override button needs to be added. Copy and paste `<button class="btn btn-default btn-sm remove cc_rfq_link_button createCreditCase" type="button">{{pageLabelMap 'CCTrainingCart_CreditOverrideBtn'}}</button>` here. Afterwards, we'll add the Javascript logic for invoking your remote action when a user requests a credit limit override.

⑤ Create a Javascript variable for your remote action.

⑥ Define your remote action by extending the `CCRZ.RemoteInvocation` Javascript object.

⑦ Gain access to the Backbone view of the Cart page via the pubsub refresh event.

⑧ Create a Javascript function for invoking the remote action in the controller.

⑨ Upon successful execution, display a message on the cart page.

⑩ Bind the click event of your button to the newly created Javascript function.

⑪ Bind the event described above to your Backbone view.

e. **Create a page label with the following values**

   i. Page Name : Cart

   ii. Page Label Name : CCTrainingCart_CreditOverrideBtn

   iii. Value : Request Override

   iv. Storefront : **DefaultStore**

f. **Create a page label with the following values**

   i. Page Name : Cart

   ii. Page Label Name : CCTraining_CreditOverrideRequested

   iii. Value : Your credit request has been sent. A member of our team will contact you once your request has been approved.

   iv. Storefront : **DefaultStore**

g. **Rebuild your page label cache**

h. **Configure CloudCraze to use your new Page Include**

   1. In your Salesforce Org, click **CC Admin** from the main tab.

   2. On the right side dropdown select the **DefaultStore** Storefront.

   3. On the left side bar, near the top, under "Settings" section, click on **Configuration Settings**.

   4. Click on **New** button to create a new Configuration Setting and enter the following to **enable** the page include:

When filling out the value for the `Page` input field referenced below, you should keep the following in mind. Start out by typing the name of the page of interest. E.g Type `Cart` and then select the appropriate value from the autocompleted suggestions in the input field.

   i. Module: Body Includes End
   ii. Configuration: Enabled
   iii. Page: Cart (CA)
   iv. Value: TRUE
5. Select **New** to create a new Configuration Setting and enter the following:
   i. Module: Body Include End
   ii. Configuration: Page Include Name
   iii. Page: Cart (CA)
   iv. Value: c__ccTrainingCartBIE

i. **Create and Activate a new Configuration Cache:**
   1. On the right side dropdown, click on the **Global Settings** link.
   2. On the left side bar, select **Configuration Cache Management**.
   3. Click on **Build New**. After a few seconds, click on **Refresh List**. A new configuration cache should have been built. Click on **"Activate"** to enable it.

j. **Verify your changes on the My Account Page**
   1. Add any value to the credit limit field on your account record
   2. Verify that the amount is displayed correctly on **My Account → Contact Information**

k. **Verify your changes on the Cart page**
   1. Add a product to the cart that exceeds your credit limit
   2. Observe that the Checkout button will no longer be visible but the newly created button is.
      i. You should see a message on the screen letting you know about the exceeded limit
   3. You'll also notice that Checkout button will be visible if the cart total doesn't exceed the credit limit.
   4. Request a credit override and observe the following
      i. A messsage should be displayed on the screen letting you know your request was successfully sent
      ii. A case should be created and associated to your cart record

Note on Remote Actions

VisualForce pages can execute functionality within the referenced controller via Remote Actions.

How do you make a remote call?

1. Reference the Apex controller in the VisualForce Page of interest.

2. Create the necessary methods in your controller. The methods should be annotated with `@RemoteAction`
3. Create a JavaScript function that extends the `CCRZ.RemoteInvocation` function.
4. Link the HTML element of interest (e.g button) to the function created in the previous step.

3.9. Order Extension Hook

3.9.1. Overview

The ccrz.cc_hk_Order extension is used to extend all CloudCraze order functionality.

Placing Orders

- place
- createTransaction
- placeTarget

Viewing Orders

- fetchOrders
- reorder

Key Points on Place Method

- Place method will be called multiple times within the order functionality
- The order place process is transactional, i.e. it is executed within a Savepoint so any updates, inserts, etc. may be rolled back if some part of the transaction fails.
- No callouts should occur within the place method.
- The same cc_hk_Order instance is used for all place calls

CloudCraze Documentation for the Order Hook

3.9.2. Create a new Order Extension

Use Case

1. Create a new Order Extension
2. Decrement the Credit Limit for the Account during the place method
3. Configure CloudCraze to use your new Order extension
4. Verify your changes

Prior to this exercise, you must create a Custom Field of type Currency on Account with the API name `CreditLimit__c`

a. **Implement your new Order Extension**

1. Create a new Apex class called `ccTraining_hk_Order` that extends `ccrz.cc_hk_Order`
2. Create a private method that accepts one argument of type `ccrz__E_Order__c`.
3. Inside your private method, query for the CreditLimit on the current order's Account.

4. Next, fetch the total amount of the current order. This is needed as the argument of type `ccrz__E_Order__c` passed into this method doesn't reference this value by default.
5. While still inside the private method, decrement the Account's Credit Limit by the current order total.
6. Update the account record
7. Override the `place` method.
8. Inside the `place` override, check if the current step is equal to `ccrz.cc_hk_Order.STEP_END`
9. If the current step is `STEP_END`, invoke the private method created in the previous step and pass in the `ccrz__E_Order__c` instance from `PARAM_ORDER` to this private method.
10. Inside the place method, return the same `Map<String,Object>` instance passed in as the method argument.

```
global without sharing class ccTraining_hk_Order extends ccrz.cc_hk_Order ①
{

  private void updateAccountCreditLimitForOrder(ccrz__E_Order__c currentOrder) ②
  {
    Account currentAccount = [SELECT CreditLimit__c FROM Account
                              WHERE Id = :currentOrder.ccrz__Account__c LIMIT 1]; ③

    ccrz__E_Order__c refetchedOrder = [SELECT ccrz__TotalAmount__c FROM ccrz__E_Order__c
                                       WHERE Id = :currentOrder.Id LIMIT 1]; ④

    Decimal orderTotal = refetchedOrder.ccrz__TotalAmount__c;
    currentAccount.CreditLimit__c -= orderTotal; ⑤

    update currentAccount; ⑥
  }


  global override Map<String, Object> place(Map<String, Object> inputData) ⑥
  {
    String currentStep = (String) inputData.get(ccrz.cc_hk_Order.PARAM_PLACE_STEP);

    if (ccrz.cc_hk_Order.STEP_END.equals(currentStep)) ⑦
     {
      ccrz__E_Order__c currentOrder =
        (ccrz__E_Order__c)inputData.get(ccrz.cc_hk_Order.PARAM_ORDER);

        ⑧
      updateAccountCreditLimitForOrder(currentOrder);
    }
    return inputData; ⑨
  }


}
```

We are updating the `Account.CreditLimit__c` field inside of a without sharing class. This allows us to keep the Account record locked down from a sharing setting perspective.

a. **Configure CloudCraze to use your new Order extension**

1. In your Salesforce Org, click **CC Admin** from the main tab.
2. On the right side dropdown select the **DefaultStore** Storefront.
3. On the left side bar, near the middle, under "Integrations" section, click on **Orders**.
4. Scroll down to the Order Hook section and set the "Order Hook API Class" to `c.ccTraining_hk_Order`

b. **Verify your changes**

1. Observe the current value of your credit limit by navigating to the My Account → Profile page.
2. Navigate to the Homepage and select or perhaps seach for any product that is within your credit limit.
3. Add the product to the cart
4. Complete the checkout process
5. Navigate to the My Account → Profile page to verify that the credit limit has since decreased.

Make sure you have `cc_hlpr_GenericHKFactory` set under the `Order Hook Factory Class.` This is required for all legacy hook extension classes to be properly insantiated inside the CloudCraze Managed Package.

3.10. Case Handling via Process Builder

Lightning Process Builder is a workflow tool that automate business processes within Salesforce.

Salesforce Help

Use Case

1. Create new attributes on the Case and CC Cart to track the requested Credit Override.
2. Create a Process Workflow to transfer the requested credit limit override amount from the Case to the Cart
3. Create a data service provider for Carts
4. Configure CloudCraze to use the new data service provider
5. Update the cart extension point to check for the credit limit override amount
6. Verify your changes

a. **Add a custom field to the Case object**

1. Create a new attribute for linking a case to its source cart on the Case object.
2. While in the setup menu, Type `Case` in the quick find box.
3. Select **Fields** under the **Build → Customize → Cases → Fields** menu.
4. Scroll down to the `Custom Fields and Relationships` section and then select `New`
5. Select `Currency` as the Data Type on the next screen
6. Set the following on the subsequent screen.
    i. Field Label: Credit Limit Override Amount
    ii. Field Name: CreditLimitOverrideAmount
    iii. Length: 16
    iv. Decimal Places: 2
    v. Leave other fields as is
7. Make the field visible for all the profiles.
8. Keep advancing through the setup screens and then save your changes.

b. **Add a custom field to the Cart object**

1. Create a new attribute for indicating the state of a credit override on the Cart Object
2. While in the setup menu, Type `Object` in the quick find box.
3. Select **Objects** under the **Build → Create → Objects** menu.
4. Select **CC Cart** from the list of the available custom objects. **Do not** select the Edit button.
5. Scroll down to the `Custom Fields and Relationships` section and then select `New`

6. Select `Currency` as the Data type on the next screen
7. Set the following on the subsequent screen.
    i. Field Label: Credit Limit Override Amount
    ii. Field Name: CreditOverride
    iii. Length: 16
    iv. Decimal Places: 2
    v. Leave other fields as is
8. Make the field visible for all the profiles.
9. Keep advancing through the setup screens and then save your changes.

c. **Create the Process Builder**

1. While in the setup menu, Type `Process builder` in the quick find box.
2. Select **Process Builder** under the **Build → Create → Workflow & Approvals → Process Builder** menu.
3. Select `New` in the Process Builder User Interface and enter the following:
    i. Process Name: Credit Case Approval
    ii. API Name: Credit_Case_Approval
    iii. Description: Process for handling cases
    iv. The process starts when: A record changes
4. Save your changes
5. Select the `+ Add Object` button and a separate window should open.
6. Enter the following on this screen.
    i. Object: Select `Case` from the dropdown options list
    ii. Start the process: When a record is created or edited
    iii. Save your changes
7. Select the `+ Add Criteria` button and a separate window should open.
8. Enter the following on this screen.
    i. Criteria Name: Override Amount Set
    ii. Criteria for Executing Actions: Conditions are met
    iii. Set Conditions **(choose from drop down menus)** :
        A. Field – Credit Limit Override Amount
        B. Operator – Does not equal
        C. Type – Global Constant
        D. Value - $GlobalConstant.Null
        E. Conditions – All of the conditions are met (AND)
        F. Save your changes

9. Select the `+ Add Action` button and a separate window should open.
10. Enter the following on this screen.
    i. Action Type: Update Records
    ii. Action Name: Set Cart Override
    iii. Record Type : Select the magnifying glass icon and enter the following:
        A. Choose the `Select a record related to the Case` option.
        B. Select `Reference Cart` from the dropdown options list
        C. Select the `Choose` button
    iv. Criteria for Updating Records: No criteria – just update records!
    v. Set new field values for the records you update:
        A. Field: Credit Limit Override Amount
        B. Type: Reference
        C. Value: Case.CreditLimitOverrideAmount
11. Save your changes
12. Click Activate to enable the process.

d. **Create a data service provider for Carts**

1. Create a new Apex class called `ccTrainingServiceCart` that extends `ccrz.ccServiceCart`
2. Override the `getFieldsMap` method.
3. Inside the `getFieldsMap` override, retreive the base fields this class returns by default
4. While still inside this method, concatenate `CreditOverride__c` to the String that contains the default base fields.
5. Return the newly created String

```
global with sharing class ccTrainingServiceCart extends ccrz.ccServiceCart ①
{
    global virtual override Map<String, Object> getFieldsMap(Map<String, Object> inputData) ②
    {
        inputData = super.getFieldsMap(inputData); ③

        String objectFields = (String)inputData.get(ccrz.ccService.OBJECTFIELDS); ③

        objectFields += ',CreditOverride__c' ; ④

        return new Map <String,Object> {ccrz.ccService.OBJECTFIELDS => objectFields}; ⑤
    }
}
```

e. **Configure CloudCraze to use your new data service Provider**

1. In your Salesforce Org, click **CC Admin** from the main tab.
2. On the right side dropdown select the **DefaultStore** Storefront.

3. On the left side bar, select "Service Management"
4. Scroll down to the data service provider section and search for ccServiceCart
5. Set the class name to `c.ccTrainingServiceCart`

f. **Update the Cart extension point to check for the Credit Limit Override Amount**

1. Search for `ccTrainingLogicCartValidate.cls` in your Integerated Developer Environment (IDE) or Developer console.
2. Inside the `checkAccountCreditLimit` method, add a new variable of type Decimal named `creditOverride`. Set the value of the credit override amount from the cart record to this new variable.
3. Add an additional check if the Cart SubTotalAmount is less than or equal to the `creditOverride` value, set `ALLOW_CHECKOUT` to true.
4. If the cart's `CreditOverride` is not null and greater or equal to the cart's totalAmount, then set isAllowCheckout to TRUE. In addition to this, you should also create a page message with the following information:
   i. labelId = 'CCTraining_CreditOverrideApproved';
   ii. type = ccrz.cc_bean_Message.MessageType.CUSTOM;
   iii. severity = ccrz.cc_bean_Message.MessageSeverity.INFO;
   iv. classToAppend = 'messagingSection-Info';

```
global with sharing class ccTraining_ccLogicCartValidate extends ccrz.ccLogicCartValidate{ ①

  private Decimal creditOverride {get; set;}

  private Boolean checkAccountCreditLimit(Map<String, Object> inputData) {
    Decimal accountCreditLimit = ccrz.cc_CallContext.currAccount.CreditLimit__c;
    List<Object> cartObjList = (List<Object>) inputData.get(ccrz.ccApiCart.CART_OBJLIST);
    ccrz__E_Cart__c cartRecord = (ccrz__E_Cart__c) cartObjList.get(0);
    creditOverride = cartRecord.CreditOverride__c; ②

    if (cartRecord.ccrz__SubTotalAmount__c <= accountCreditLimit
        || cartRecord.ccrz__SubTotalAmount__c <= creditOverride) { ③
      return true;
    } else {
      return false;
    }
  }

  global override Map<String, Object> processValidate(Map<String, Object> inputData) {
    Map<String, Object> cartFetchRequest = new Map<String, Object>(inputData);
    cartFetchRequest.put(ccrz.ccApi.SIZING,
        new Map<String, Object> {
            ccrz.ccApiCart.ENTITYNAME => new Map<String, Object> {
                ccrz.ccApi.SZ_DATA => ccrz.ccApi.SZ_L // LARGE is the smallest size OOTB that will
return for us SubtotalAmount__c
                ,ccrz.ccApi.SZ_ASSC => FALSE // for this override we are only interested in total
amount and not any details of individual line items
                ,ccrz.ccApi.SZ_SKIPTRZ => TRUE // skip transform so, we can work directly with the
sObject record here in the backend.
            }
```

```
        }
    );

    Map<String, Object> cartFetchResponse = ccrz.ccApiCart.fetch(cartFetchRequest);
    Boolean shouldAllowCheckout = checkAccountCreditLimit(cartFetchResponse);

    List<ccrz.cc_bean_Message> retMessages = new List<ccrz.cc_bean_Message>();

    if(!shouldAllowCheckout) {
      ccrz.cc_bean_Message msg = new ccrz.cc_bean_Message();
      msg.labelId              = 'CCTraining_CreditExceeded';
      msg.type                 = ccrz.cc_bean_Message.MessageType.CUSTOM;
      msg.severity             = ccrz.cc_bean_Message.MessageSeverity.ERROR;
      msg.classToAppend        = 'messagingSection-Info';

      retMessages.add(msg);
    } else if (creditOverride > 0.0) { ④
      ccrz.cc_bean_Message msg = new ccrz.cc_bean_Message();
      msg.labelId              = 'CCTraining_CreditOverrideApproved';
      msg.type                 = ccrz.cc_bean_Message.MessageType.CUSTOM;
      msg.severity             = ccrz.cc_bean_Message.MessageSeverity.INFO;
      msg.classToAppend        = 'messagingSection-Info';

      retMessages.add(msg);
    }

    inputData.put(ccrz.ccApiCart.ALLOW_CHECKOUT, shouldAllowCheckout);
    inputData.put(ccrz.ccApi.MESSAGES, retMessages);

    return inputData;
  }
}
```

g. **Create a page label with the following values**

   i. Page Name : Cart

   ii. Page Label Name : CCTraining_CreditOverrideApproved

   iii. Value : Your credit request has been approved.

   iv. Storefront : **DefaultStore**

      1. Rebuild your page label cache

h. **Verify your changes**

3.11. Invoice Extension Hook

- CloudCraze provides invoice capabilities which is essentially just billing. i.e. collecting payment outside the checkout process.
- Invoices can be created during the order creation process for orders and they can also be created independent of an order.
- The extension point we will edit for this business case is `ccrz.cc_hk_Invoice`.

3.11.1. Method Overview

Invoice creation

- **onOrder** – Method called during order placement between the `cc_hk_Order` steps of `STEP_INVOICE_CREATE_PRE` and `STEP_INVOICE_CREATE_POST`.
  - This method will only work for signed in users and it creates the Invoice object.

View the Invoice

- **fetchInvoice** - The fetchInvoice method is used by the invoice detail page to fetch the invoice data for display.
- **fetchInvoices** - The fetchInvoices method is used by the MyAccount → My Invoices page to return the set of invoices to be shown.
- **fetchInvoicesForPayment** - The fetchInvoicesForPayment method is used by the invoice payment screen to determine the set of invoices that will be paid against.

Pay an Invoice

- **validatePayment** – Called when the user enters a payment against one or more. invoices on the invoice payment screen. Callouts should be made from this method.
- **parseFetchPaymentData** - Parses passed payment data from the user interface.

CloudCraze Documentation for the Invoice Hook

3.11.2. Exercise: Create a new Invoice Extension

Use Case

1. Create a new Invoice Extension
2. Update the Credit Limit amount based on Invoice payments
3. Configure CloudCraze to use your new Invoice extension

Prior to this exercise, you must create a custom field of type Currency on Account with the API name `CreditLimit__c`. Ignore this step if you've created this field in a prior exercise.

a. **Implement your new Invoice Extension**

1. Create a new Apex class called `ccTraining_hk_Invoice` that extends `ccrz.cc_hk_Invoice`
2. Override the `applyPayment` method.
3. Invoke the base implementation (super) of the `applyPayment` method first. This is needed because we would like to add to the credit **after** the payment has been processed.
4. After invoking the base implemementation of `applyPayment`, access the deserialized invoice payment data as a `Map<String,Object>`. It is worth noting that the keys are the invoice Salesforce id's while the values are payment amounts formatted as Strings.
5. Create a local variable to store the total dollar amount paid.
6. Iterate through the list of payments and add all the decimal values of the payments from `ccrz.cc_hk_Invoice.PARAM_INVOICE_PAYMENTS`.
    i. These should be rounded to two decimal points using the HALF_EVEN rounding method at each step.
7. Leverage `ccrz.cc_CallContext.currAccountId` to query for the Credit Limit associated with the current user's account.
8. Increase the Account's Credit Limit by the amount paid for the current invoice.
9. Return the `Map<String,Object>` data received from the call to `super.applyPayment`

```
global without sharing class ccTraining_hk_Invoice extends ccrz.cc_hk_Invoice ①
{
  global override Map<String, Object> applyPayment(Map<String, Object> inputData) ②
  {
    Map<String, Object> retData = super.applyPayment(inputData); ③

    Map<String, Object> invoicePaymentMap =
      (Map<String, Object>)inputData.get(ccrz.cc_hk_Invoice.PARAM_INVOICE_PAYMENTS); ④

    Decimal totalAmountPaid = 0.0; ⑤

    for (Object paymentAmountObject : invoicePaymentMap.values()) ⑥
     {
      Decimal paymentAmountVal = Decimal.valueOf((String)paymentAmountObject);
      totalAmountPaid += paymentAmountVal.setScale(2,System.RoundingMode.HALF_EVEN); ⑥
    }

    Account currentAccount = [SELECT CreditLimit__c FROM Account
              WHERE Id = :ccrz.cc_CallContext.currAccountId LIMIT 1]; ⑦

    currentAccount.CreditLimit__c += totalAmountPaid; ⑧

    update currentAccount;

    return retData; ⑨
  }
}
```

We are updating the `Account.CreditLimit__c` field inside of a without sharing class. This allows us to keep the Account record locked down from a sharing setting perspective.

a. **Configure CloudCraze to use your new Invoice extension**

1. In your Salesforce Org, click **CC Admin** from the main tab.
2. On the right side dropdown select the **DefaultStore** Storefront.
3. On the left side bar, near the middle, under "Integrations" section, click on **Orders**.
4. Scroll down to the Invoice Hook section and set the "Invoice Hook API Class" to `c.ccTraining_hk_Invoice`

Make sure you have `cc_hlpr_GenericHKFactory` set under the "Invoice Hook Factory Class." This is required for all legacy hook extension classes to be properly insantiated inside the CloudCraze Managed Package.

3.12. Payment

Adding a new payment type is a combination of configuration and development that involves the following:

1. Configure the new payment type in the Configuration Module and Settings menu
2. Create VisualForce Pages and Apex Classes for the new payment type functionality.

CloudCraze, by default has an existing set of configurations for purchase orders. It's encouraged for suscribers to use this as a reference when implementing the functionality for a new payment type.

Configuration Module (Purchase Order)

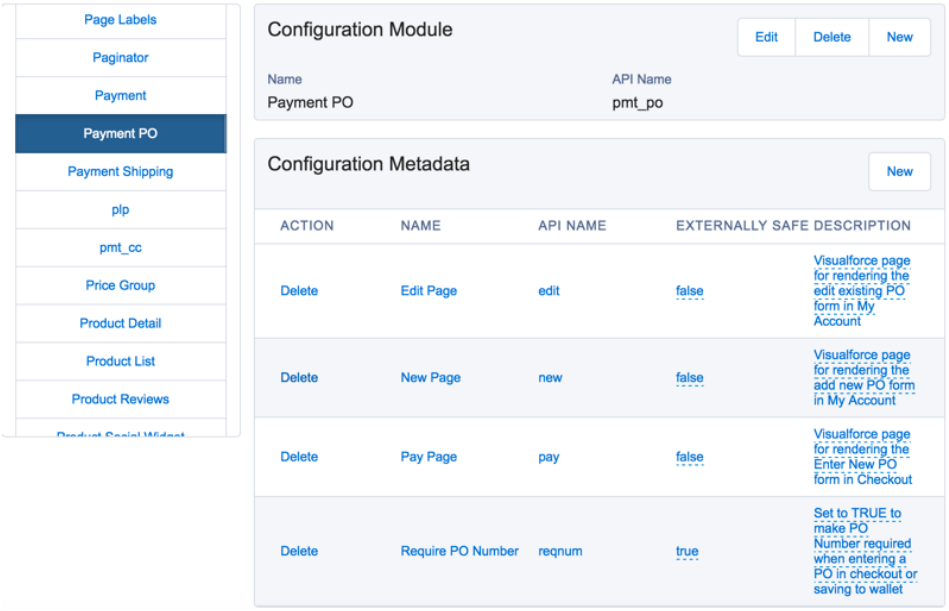


Figure 3. Purchase order configuration

You can access the above by navigating to **CC Admin → Configuration Modules** and then selecting **Payment PO**.

Note that the configuration modules options is not storefront specific and hence only accessible at the `Global Settings` level.

Configuration Setup

The corresponding Configuration Settings for the module described above is as follows.

Pay (pmt_po.pay)

Configuration setting that indicates the VisualForce Page leveraged for processing Purcahse Order payment type in the Checkout flow.

Edit (pmt_po.edit)

Configuration setting that indicates the VisualForce Page leveraged for editing existing Purcahse Order payment information via the My Account page.

New (pmt_po.new)

Configuration setting that indicates the VisualForce Page leveraged for saving new Purchase Order payment information via the My Account page

Since these settings are provided by default in CloudCraze, they are global and can hence be accessed as follows

Navigate to **CC Admin → Configuration Setting** and then select **Payment PO** from the dropdown options for `Module`.

Conversely, this can also be accessed at the storefront level if the `Display Global` field on the configuration settings page is left checked.

An overview of the signifance of these configurations is available here

3.12.1. Development

The payment integration varies for most clients. However, the expectation in CloudCraze is to trigger a payment event upon successful completion of the callout. This payment event accepts a payload which typically contains the information you'll be needing to persist e.g (card token). This event is shown in the image below.

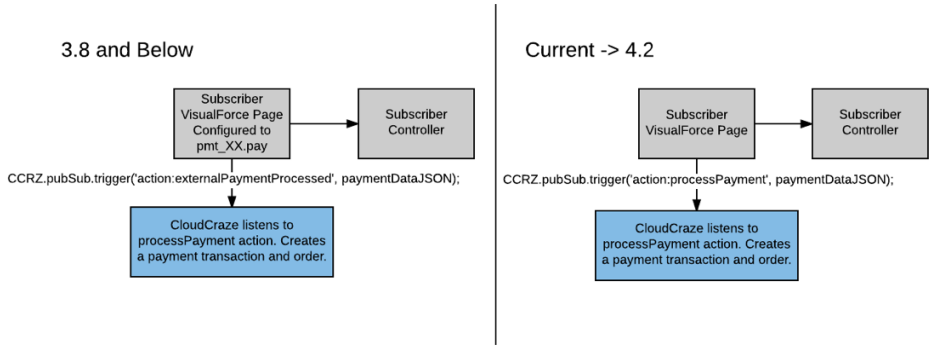


Figure 4. Credit Card configuration

3.12.2. Exercise: Configure a new payment type

Use Case

1. Create a configuration module for the new payment type
2. Create a VisualForce page for the new payment type
3. Create a configuration setting for the VisualForce page

a. **Create a Configuration Module for the new Payment type**

1. In your Salesforce Org, select **CC Admin** from the main tab.
2. On the right hand side in the menu bar, select **{Global Settings}** - This should be preset for you by default upon page load.

3. Select **Configuration Modules** secfrom the menu on the left hand side.
    i. As previously mentioned, we'll want to use th existing setup as a reference for our new type.
    ii. You can open a separate browser window with the existing settings for purchase orders to verify your work.
4. Select the **New** button in the **Configuration Module** section and fill out the following and save the changes:

    Name

    Payment CC

    API Name

    pmt_cc

5. Select the **New** button in the **Configuration Metadata** section and fill out the following:
    i. Metadata for **Pay** page

    Name

    Pay page

    API Name

    pay

    Description

    VisualForce Page leveraged for processing Credit Card payments in the Checkout flow

    Externally Safe

    FALSE **(Leave it unchecked)**

    ii. Metadata for **New** page

    Name

    New page

    API Name

    new

    Description

    VisualForce Page leveraged for saving new Credit Card payment information via the My Account page

    Externally Safe

    FALSE **(Leave it unchecked)**

    iii. Metadata for **Edit** page

    Name

Edit page

API Name

edit

Description

VisualForce Page leveraged for editing existing credit card payment information via the My Account page

Externally Safe

FALSE **(Leave it unchecked)**

b. **Create a VisualForce page for processing credit card payments for the Checkout flow**

1. Create a new VisualForce page named `ccTrainingPaymentCC`

   i. In your SalesForce Org, click the **Developer Console** link under the dropdown menu **under your name** - this should be on the top right hand corner on the page.

   ii. A new window should pop-up.

   iii. In the Developer Console menu, select: **File → New → VisualForce Page**

   iv. In the popup window (New Apex Page), enter **ccTrainingPaymentCC** as your page name and click **ok**.

   v. Remove the boilerplate code (delete the contents of the page).

   vi. Enter the following: note that the code below was added for your convenience. As previously mentioned you can use the existing pages for purchase orders as a starting point. The VisualForce pages for the purchase order payment type are available in your Salesforce org by default.

```
<apex:page applyHtmlTag="false" docType="html-5.0" sidebar="false" showHeader="false"
standardStylesheets="false" cache="false">

<script id="PaymentCC-Both" type="text/template"> ①
    <div class="ccPaymentOuterContainer">
        <h2 class="title">{{pageLabelMap 'MyWallet_AddPymtMtd'}}</h2>
        <div class="main_content_large_right_container">
            <div class="alert alert-error ccPayment-messagingSection-Error" style="display:
none"></div>
            <form id="newCCForm" action="" class="newCCForm" forceSSL="true">
            <fieldset>
                <legend>New Credit Card</legend>
                <p>Enter your payment data below</p>
                <p>Card Type:</p>
                <p><select name="paymentType">②
                <option value="Visa">Visa</option>
                <option value="Mastercard">Master Card</option>
                <option value="Amex">American Express</option>
                <option value="Discover">Discover Card</option>
                </select></p>
                <p>Card Number:</p><p><input type="text" name="accountNumber"/></p> ③
                <p>Expiration:</p><p>M:<select name="expirationMonth"> ④
                    <option value="01">01</option>
                    <option value="02">02</option>
```

```
                    <option value="03">03</option>
                    <option value="04">04</option>
                    <option value="05">05</option>
                    <option value="06">06</option>
                    <option value="07">07</option>
                    <option value="08">08</option>
                    <option value="09">09</option>
                    <option value="10">10</option>
                    <option value="11">11</option>
                    <option value="12">12</option>
                </select>
                 Y:<select name="expirationYear"> ⑤
                    <option value="2015">2015</option>
                    <option value="2016">2016</option>
                    <option value="2017">2017</option>
                    <option value="2018">2018</option>
                    <option value="2019">2019</option>
                    <option value="2020">2020</option>
                </select></p>
            </fieldset>
                <p class="two_buttons">
                    <div class="right">
                        <input type="button" class="button makeCCPayment" id="btnMakeCCPayment"
data-id="newCCForm" value="{{pageLabelMap 'CCTrainingPayment_MakePayment'}}" /> ⑥
                        <input type="hidden" name="accountType" value="cc"/> ⑦
                    </div>
                </p>
            </form>
        </div>
    </div>
</script>

<script>
          ⑧
    jQuery(function($)
    {
        CCRZ.models.PaymentsCCModel = CCRZ.CloudCrazeModel.extend(); ⑨

        CCRZ.views.PaymentsCCView = CCRZ.CloudCrazeView.extend({ ⑩
            viewName : "PaymentsCCView",
            managedSubView : true,
            templateDesktop : CCRZ.util.template("PaymentCC-Both"), ⑪
            templatePhone : CCRZ.util.template("PaymentCC-Both"), ⑪
            init : function(options)
            {
                this.selector = options.selector;
                this.render();
                CCRZ.pubSub.trigger('action:paymentViewInit',this);
            },
            events:
            {
                "click .makeCCPayment" : "makeCCPayment" ⑫
            },
            validateInfo: function(formName, fieldData)
            {
                $("#"+formName).validate({
                    invalidHandler: function(event, validator)
                    {
                        CCRZ.handleValidationErrors(event, validator, 'ccPayment-
```

```
messagingSection-Error', false);
                    },
                    rules:
                    {
                        accountNumber : { required: true, minlength: 4 }
                    },
                    messages:
                    {
                        accountNumber : { required : 'Card number is required', minlength:
'Card number is not valid' }
                    },
                    errorPlacement: function(error, element) {
                    }
                });
                return $("#"+formName).valid();
            },

            parentPreRender : function()
            {
                //Look for the instantiated iframe
                //Detach it from the DOM
                //Re-render the view but don't create the iframe
                //Re-attach the iframe as appropriate in the rendered view
            },
            renderDesktop : function()
            {
                this.setElement(this.selector);
                this.data={};
                this.$el.html(this.templateDesktop(this.data));
            },
            renderPhone : function()
            {
                this.setElement(this.selector);
                this.data={};
                this.$el.html(this.templatePhone(this.data));
            },
            makeCCPayment : function(event) ⑬
            {
                var formName = $(event.target).data("id");
                var formData = form2js(formName, '.', false, function(node) {}, false);
                if (this.validateInfo(formName, formData))
                {
                    var paymentData = formData;
                    paymentData.token = 'ABCD123456789'; ⑭
                    paymentData.accountNumber = '';  ⑮

                    CCRZ.pubSub.trigger('action:processPayment', paymentData); ⑯
                }
            }
        });
        CCRZ.pubSub.trigger('action:paymentViewReady','cc',function(options)
        {
            CCRZ.payment = CCRZ.payment||{views:{}};

            CCRZ.payment.views.cc = new CCRZ.views.PaymentsCCView({
                model : new CCRZ.models.PaymentsCCModel(), ⑰
                selector : options.selector
            });
        });
```

```
    });
</script>
</apex:page>
```

① Handlebars template ID defined here should match the value referenced in the **templateDesktop** and **templatePhone** properties in the Javascript section on this page.

② List of payment options. This is hard-coded here for convenience. In practice, you'll ideally want to save these values in a custom metadata type and subsequently fecth them via the logic in your Apex controller.

③ Input field for Credit Card number

④ List of expiration months. This is hard-coded here for convenience. In practice, you'll ideally want to save these values in a custom metadata type and subsequently fecth them via the logic in your Apex controller.

⑤ List of expiration years. This is hard-coded here for convenience. In practice, you'll ideally want to save these values in a custom metadata type and subsequently fecth them via the logic in your Apex controller.

⑥ Button that invokes the payment processing logic.

⑦ Input field for the **accountType**. This is set as **cc** which means credit card. Alternatively, it can be set in the Javascript function described in the step above.

⑧ Javascript section

⑨ Backbone model leveraged on this page.

⑩ Backbone view leveraged on this page.

⑪ Template ID described in step 1 is referenced here

⑫ Logic that binds the click event of the button to the Javascript function that initiates the payment process.

⑬ Javascript function invoked by the **process payment** button.

⑭ Credit card token or subscription ID. In practice, this value will be provided by the payment gateway

⑮ We are setting this value to a blank field to avoid having to save the credit card number in the org

⑯ This event initiates the order creation process. It's also responsible for creating an audit trail **(Payment transaction)** of how the user paid for the order.

⑰ Logic that associates the Backone view to its corresponding model.

c. **Create a page label with the following values**

   i. Page Name : All

   ii. Page Label Name : CCTrainingPayment_MakePayment

   iii. Value : Process Payment

   iv. Storefront : **DefaultStore**

      1. Rebuild your page label cache

d. **Create Configuration settings for the VisualForce page of the new Payment type** Since our intent is to

make our settings storefront specific, let's do the following.

1. In your Salesforce Org, select **CC Admin** from the main tab.
2. Change the selection from `Global Settings` to `DefaultStore`.
    i. This is accessible in the dropdown options list on the right hand side.
3. Select the **Configuration Settings** menu from the menu on the left side.
4. Click on **New** button to create a new Configuration Setting and enter the following:

    Module

    Payment CC

    Configuration

    Pay Page

    Page

    all

    Attribute Value

    c__ccTrainingPaymentCC

e. **Add the new VisualForce page to the list of acceptable pages for payment processing (White Listing)** While still in the storefront specific configuration settings in **CC Admin**, do the following:

1. Select **Payment** from the the dropdown list for **Module**
    i. This should filter the list of configuration settings.
2. Search for the configuration setting named **White List** from this list.
3. Select the corresponding **Override** link for this setting - this should be on the right hand side.
4. Enter the following in the popup window

    Attribute Value

    Add `,c__ccTrainingPaymentCC` at the end of existing value.

5. Leave other fields as is and save your changes.

f. **Update the list of acceptable payment types** While still in the storefront specific configuration settings in **CC Admin**, do the following:

1. Select **Checkout** from the the dropdown list for **Module**
    i. This should filter the list of configuration settings.
2. Search for the configuration setting named **Payment Types** from this list.
3. Select the corresponding **Override** link for this setting - this should be on the right hand side.
4. Enter the following in the popup window:

    Attribute Value

    Add `,cc` at the end of the existing value. The new value should be `po,cc`

5. Leave other fields as is and save your changes.

g. **Create and Activate a new Configuration Cache**

   i. Select **Global Settings** as described in the previous steps above.

   ii. Select **Configuration Cache Management** from the menu on the left hand side.

   iii. Select **Build New**, wait for a few seconds and then select **Refresh List**.

h. A new configuration cache should have been built. Click on **Activate** to enable it.

i. **Verify your Changes**

   i. Go through the checkout flow and place an order with the newly configured payment page

Chapter 4. Appendix

4.1. Setting up a new Cloudcraze Org

- Besides the configuration settings, any storefront will also require information about their products, price lists, promotions, etc. etc.
- There are multiple ways of uploading this data to a CloudCraze Org. One of them is to use is the CC Admin's Data Loader.

Real-world data uploads should be done with traditional tools like the SalesForce's DataLoader, SFDX or Third Party services.

4.1.1. The CC Admin's Data Loader

Loading data via the SalesForce Dataloader is straight-forward, but also time consuming. In order to speed up that process, CloudCraze provides a tool under **CC Admin** for Data loading purposes.

4.1.1.1. Exercise: Using the CC Admin Data Loader to add storefront info

Use Case

1. Reset the Cloudcraze Org (remove the configuration settings and any existing data)
2. Load sample CloudCraze configuration settings
3. Load sample CloudCraze storefront data

Which personas should go through this exercise?

CC Administrators, CC Developers, CC Architects

What will this exercise teach you?

How to add the default CloudCraze data via the CC Admin

Why is it important?

The default sample data provided by the OOTB CloudCraze is very useful for quick testing and troubleshooting, and it helps understand the different data objects used in the StoreFronts.

Pre-requisites

none

What will we be working with in this exercise?

CC Admin

Cloudcraze Objects

none

4.1.1.1.1. Working with the CC Admin's Data Loader

1. Reset the CloudCraze Org:
    a. In your Salesforce Org, click **CC Admin** from the main tab bar.
    b. On the left side bar near the top, select **Data Loader**.
    c. At the bottom of the page, click on **Delete** button.
    d. Confirm the deletion by clicking on **Yes, Delete All** on the modal window.
    e. If prompted again, click **Ok** on the popup window to confirm the action
    f. After a few seconds, all the CloudCraze data (including configuration settings) should have been deleted from the SalesForce org.
2. Load Sample CC Configuration Settings:
    a. In your Salesforce Org, click **CC Admin** from the main tab bar.
    b. On the left side bar near the top, select **Data Loader**.
    c. Under '"Enter the static resource containing the custom setting data"', enter **'CC_CustomSettings'** (alternatively, you can leave this field blank to take the default value).
    d. Click on **Initialize**.
    e. After a few seconds, the page will refresh, and extra options on the left side bar should have appeared.
3. Load Sample CC StoreFront Data:
    a. In your Salesforce Org, click **CC Admin** from the main tab bar.
    b. On the left side bar near the top, select **Data Loader**.
    c. In the '"CloudCraze Data"' section, under '"Enter the static resource containing the custom data"', enter **CC_Data2** (alternatively, you can leave this field blank)
    d. Click on **Load**.
    e. After a few seconds, the page will refresh. Once the Load button turns blue again, it means the StoreFront data is ready. That means that sample Products, Price Lists, Images, etc. have been loaded to the Storefront.
4. Enable Storefront theme.
    a. Configure the Default Store **Theme** (enable ccrz__CC_Theme_CloudBurstRD for the Default Store).
        i. Navigate to the CC Admin page.
        ii. Click the drop down arrow in the top right corner next to the **Global Settings** button and select DefaultStore.
        iii. In the DefaultStore settings menu on the left, select **Themes** and enable the **ccrz_CC_Theme_CloudBurstRD** theme.
    b. Create and Activate a new Configuration Cache.
        i. Navigate to the CC Admin page.
        ii. In the Global Settings menu on the left, Select **Configuration Cache Management**.
        iii. Select the **Build New** button on the top right and activate the new cache.

c. Update the account **(e.g. PortalAccount)** with the **PortalAccount Account Group**.

d. **Associate** a contact **(e.g Jon Amos)** to the account above **(e.g. PortalAccount)**.

e. Enable the customer user for the contact and set its **Profile** to **CloudCraze Customer Community User**.

f. Change the contact's email address to yours.

g. Verify your changes by logging into the Community as a user.

F.A.Q.

What are the "CC_CustomSettings" and "CC_Data2" files?

They are special static resource files that contains a collection of XML and JSON files with StoreFront settings and data.

Can I create my own Settings and Data files?

Absolutely! However, we encourage our clients to rely on SFDC or SFDX for real-world scenarios.

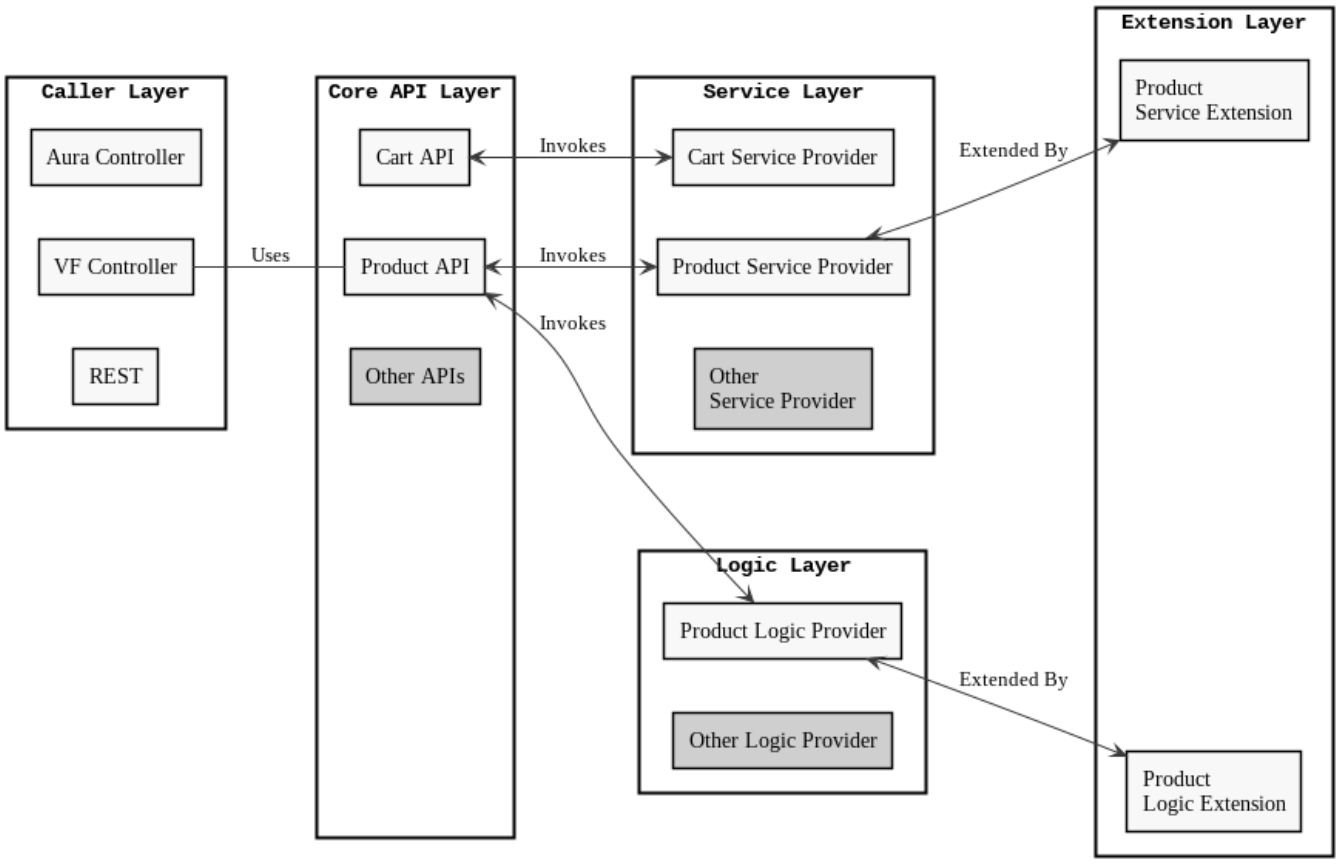
Which CloudCraze objects are included in CC_CustomSettings and CC_Data2 files?

CC_CustomSettings and CC_Data2 are static resource package types that allow you to batch load information and data for CC for your SF org. At its core, they are XML files divided by row tags, where each row represents a record to be created. The XML filename maps to the CloudCraze SalesForce object to create.

In addition to the XML files, CC_CustomSettings also include some json files (e.g. configuration-latest.json), that sets all the Configuration Settings (Global and per Store).

Remember to create and activate a new **Configuration Cache** and refresh the **Category Tree Cache**, **Menu Cache**, **Product Specs** and **Page Label Cache** indexes after uploading data to your storefronts.

4.2. CloudCraze Architecture



4.3. CloudCraze and Backbone

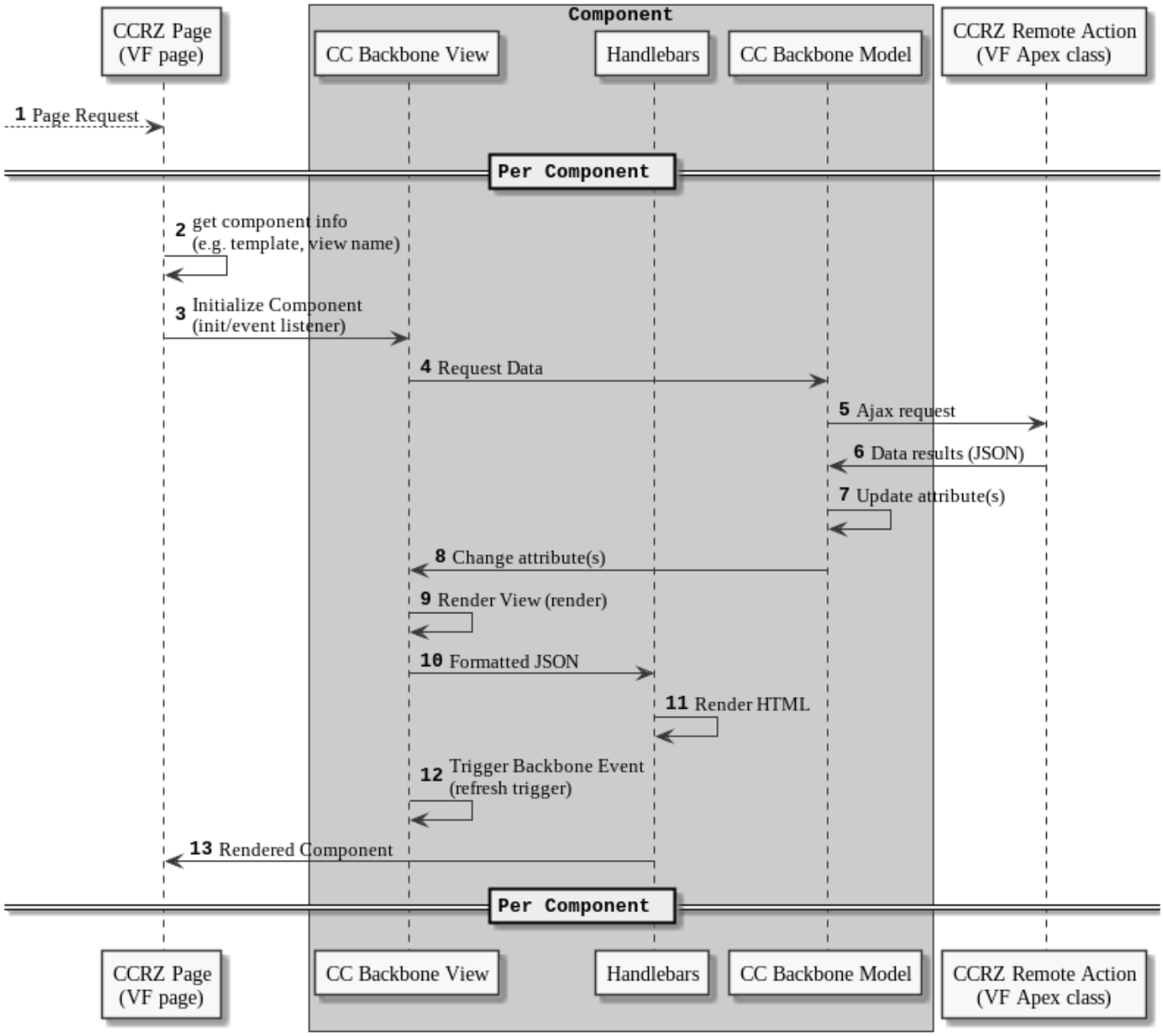


Figure 5. How Backbone Works in Cloudcraze

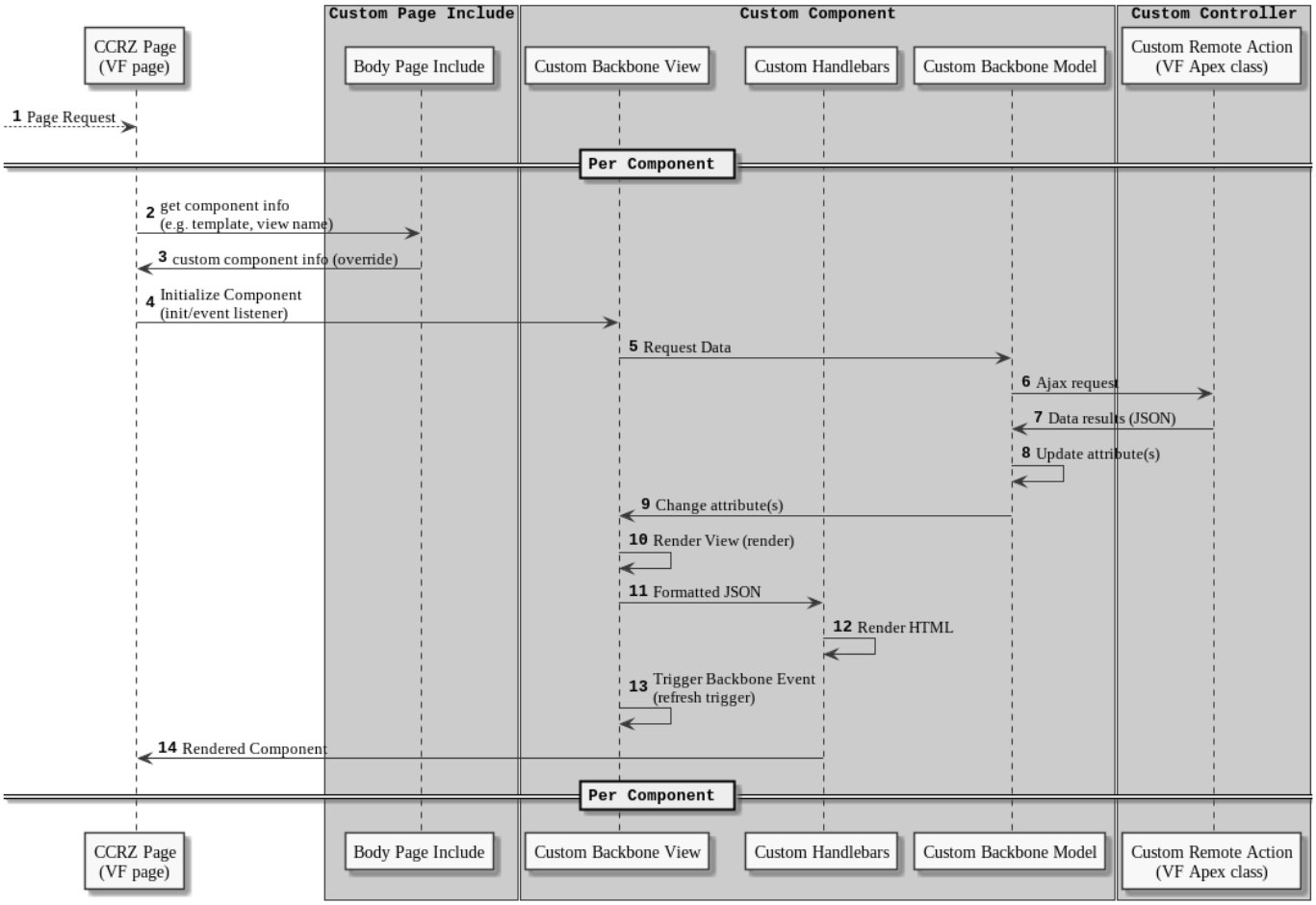
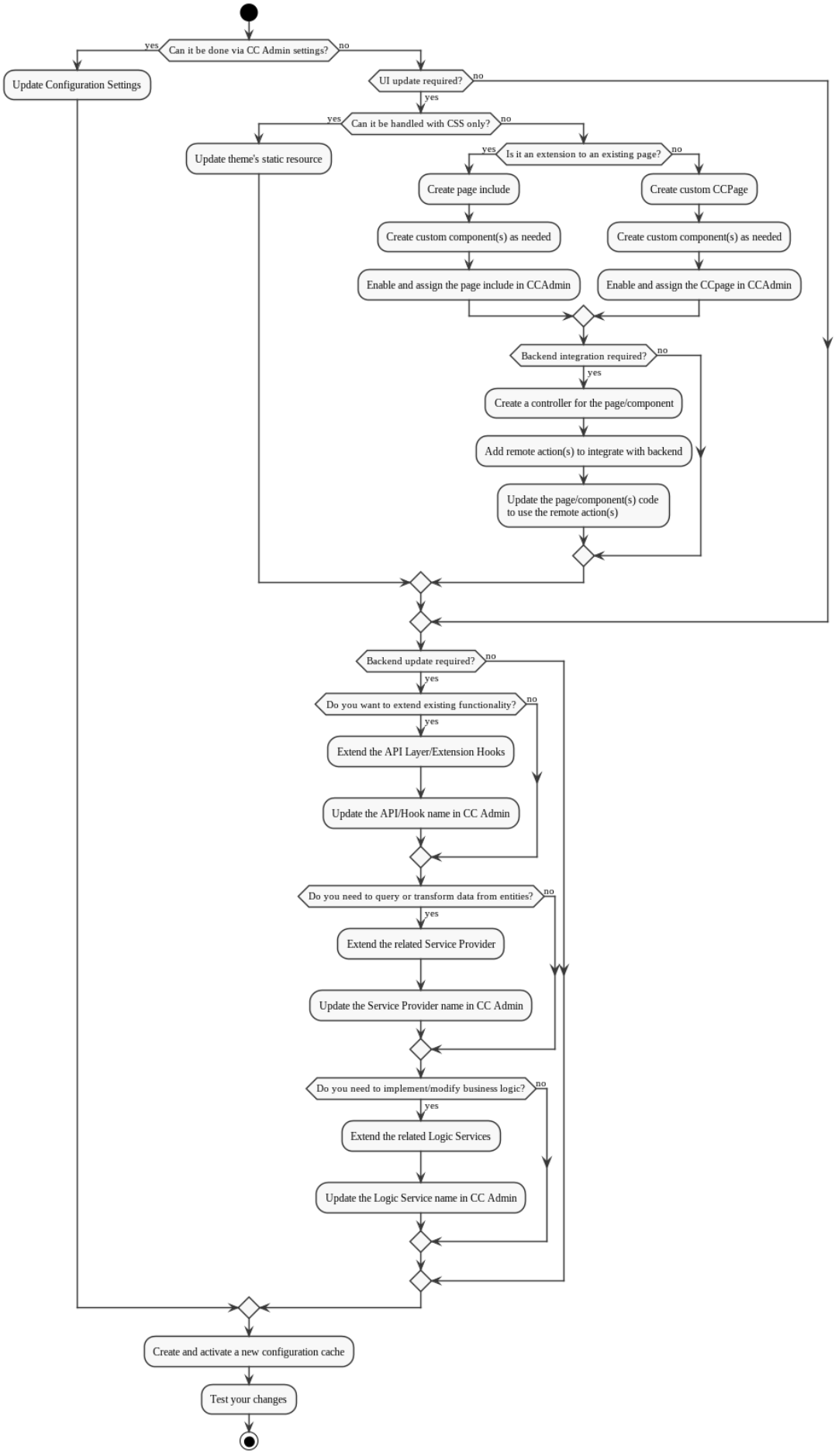


Figure 6. Extending the Cloudcraze View

4.4. Customization Tree


Can it be done via CC Admin settings?
yes
no
Update Configuration Settings
UI update required?
no
yes
Can it be handled with CSS only?
yes
no
Update theme's static resource
Is it an extension to an existing page?
yes
no
Create page include
Create custom CCPage
Create custom component(s) as needed
Create custom component(s) as needed
Enable and assign the page include in CCAdmin
Enable and assign the CCpage in CCAdmin
Backend integration required?
no
yes
Create a controller for the page/component
Add remote action(s) to integrate with backend
Update the page/component(s) code to use the remote action(s)
Backend update required?
no
yes
Do you want to extend existing functionality?
no
yes
Extend the API Layer/Extension Hooks
Update the API/Hook name in CC Admin
Do you need to query or transform data from entities?
no
yes
Extend the related Service Provider
Update the Service Provider name in CC Admin
Do you need to implement/modify business logic?
no
yes
Extend the related Logic Services
Update the Logic Service name in CC Admin
Create and activate a new configuration cache
Test your changes

4.5. Order Hook Override

Order extension to demonstrate the many steps involved in the order placement and how ccLog works. To test, place an order with the ccLog enabled in the URL

```
global with sharing class ccTraining_hk_OrderSteps extends ccrz.cc_hk_Order { ①
  global override Map<String, Object> place(Map<String, Object> inputData) { ②
    String currentStep = (String) inputData.get(ccrz.cc_hk_Order.PARAM_PLACE_STEP);

    ccrz.ccLog.log(System.LoggingLevel.Info,
                  'M:E:ccTraining_hk_Order:place:currentStep',
                  currentStep); ③

    if (ccrz.cc_hk_Order.STEP_END.equals(currentStep)) { ④
      ccrz__E_Order__c currentOrder =
        (ccrz__E_Order__c)inputData.get(ccrz.cc_hk_Order.PARAM_ORDER);

        ccrz.ccLog.log(System.LoggingLevel.DEBUG,
                      'M:X:ccTraining_hk_Order:place:currentOrder',
                      currentOrder);

    }
    return inputData; ⑤
  }
}
```

① Create an extension of ccrz.cc_hk_Order

② Override the place method

③ Everytime the method is called, get the current step name and log it into ccLog

④ Additionally, if it is the last step, get the order data and print it out into ccLog

⑤ Passthrough the inputdata back

4.6. Catalog Hook Override

Catalog extension to demonstrate how to extend the catalog filtering and how ccLog works. To test, place an order with the ccLog enabled in the URL

```
global with sharing class ccTraining_hk_Catalog extends ccrz.cc_hk_Catalog {
  global override Map<String, Object> filterCatalogData(Map<String, Object> inputData) {

    ccrz.ccLog.log(System.LoggingLevel.Info,
                'M:E:ccTraining_hk_Catalog:filterCatalogData'); ①

    List<String> filteredSkus = new List<String>();
    for(ccrz__E_Product__c p : [
      select
        ccrz__SKU__c
      from
        ccrz__E_Product__c
      where
        ccrz__ProductType__c = 'Product' ②
        and ccrz__SKU__c in
          :(List<String>)inputData.get(ccrz.cc_hk_Catalog.PARAM_SKU_LIST)
    ])
    {
      filteredSkus.add(p.ccrz__SKU__c);
    }

    ccrz.ccLog.log(System.LoggingLevel.Info,
                'M:X:ccTraining_hk_Catalog:filterCatalogData:filteredSkus',
                filteredSkus); ③

    return new Map<String,Object>{
        ccrz.cc_hk_Catalog.PARAM_SKU_LIST => filteredSkus
    };
  }

  global override Map<String, Object> autoComplete(Map<String, Object> inputData) {
    String searchString = (String) inputData.get(ccrz.cc_hk_Catalog.PARAM_SEARCH_STRING);

    ccrz.ccLog.log(System.LoggingLevel.Info,
                'M:E:ccTraining_hk_Catalog:autoComplete:searchString',
                searchString); ④

    System.debug('ccTraining_hk_Catalog:autoComplete:string to search ='
                  + searchString); ⑤

    return super.autoComplete(inputData);
  }
}
```

① We can use ccLog to confirm the function is getting called.

② Filter the list of SKUs to only return the products of type "Product" (exludes Bundles and Kits).

③ We can use ccLog to view the new list of SKUs that will be returned.

④ Sometimes ccLog is disabled (e.g. for autoComplete and modifyAutoComplete methods)

⑤ In those cases you can still use SalesForce's methods (e.g. System.debug)

Handlebars Overrides:

- Handlebars.js live editor

Managing your Data in SalesForce:

- DataLoader CLI
- Jenkins and Dataloader.jar
- SFDX
- Heroku Connect
- DBAmp
- How to upgrade and patch
- Deploying code, custom themes
- SalesForce limits

Known issues: - Preventing Standard Salesforce pages on Communities