SFCC B2C Cartridge Coding Style Guide

Table of Contents

1. Introduction

This coding style guide provides comprehensive standards and best practices for developing Salesforce Commerce Cloud (SFCC) B2C cartridges. Following these guidelines ensures consistent, maintainable, secure, and high-performance code across development teams.

1.1 Purpose

- Establish consistent coding standards across development teams
- Improve code readability, maintainability, and scalability
- Ensure security best practices are followed
- Facilitate collaboration between developers
- Reduce bugs and improve overall code quality

1.2 Scope

This guide applies to all SFCC B2C cartridge development using the Storefront Reference Architecture (SFRA) framework.

2. Project Structure and Organization

2.1 Cartridge Structure

Follow the standard SFCC cartridge structure:

```
app_custom_cartridge/
├── cartridge/
│   ├── client/
│   │   └── default/
│   │       ├── js/
│   │       └── scss/
│   ├── controllers/
│   ├── models/
│   ├── scripts/
│   │   ├── helpers/
│   │   └── util/
│   ├── templates/
│   │   └── default/
│   └── config/
├── webpack.config.js
└── package.json
```

2.2 Cartridge Naming Conventions

- **Generic cartridges**: Use descriptive names like `app_storefront_base`
- **Application-specific cartridges**: Use format `app_custom_[client_name]`
- **Plugin cartridges**: Use format `plugin_[service_name]`
- **Integration cartridges**: Use format `int_[integration_name]`

2.3 File Organization

- Group related functionality in logical directories
- Use consistent folder structures across cartridges
- Separate client-side and server-side code clearly
- Place reusable utilities in dedicated folders

3. Naming Conventions

3.1 General Naming Rules

- Use **PascalCase** for constructors and classes
- Use **camelCase** for variables, functions, and methods
- Use **UPPER_CASE** for constants
- Use **kebab-case** for CSS classes and file names
- Use **snake_case** for database-related names

3.2 JavaScript Variables and Functions

```
// Variables - camelCase
var productPrice = 100;
var customerData = {};

// Functions - camelCase with descriptive verbs
function calculateTotalPrice() {}
function validateEmailAddress() {}

// Constants - UPPER_CASE
var MAX_QUANTITY = 99;
var DEFAULT_LOCALE = 'en_US';

// Constructors/Classes - PascalCase
function ProductModel() {}
```

3.3 ISML Templates

```
product-details.isml
checkout-confirmation.isml


<div>
    <div>
        <div>
```

3.4 Controller Naming

```
// Controller files: PascalCase.js
Product.js
Cart.js
CustomerAccount.js

// Route names: descriptive and consistent
```

```
server.get('Show', function (req, res, next) {});
server.post('AddToCart', function (req, res, next) {});
```

4. JavaScript Coding Standards

4.1 General JavaScript Standards

```
'use strict';

// Use strict mode in all JavaScript files
// Always declare variables with var, let, or const
// Prefer const for constants, let for block-scoped variables

// Good
const API_ENDPOINT = 'https://api.example.com';
let productCount = 0;
var customerSession = require('dw/system/Session').customer;

// Avoid
MAX_ITEMS = 100; // Missing declaration
```

4.2 Function Declarations

```
// Function declarations - use descriptive names
function calculateShippingCost(product, shippingMethod) {
    if (!product || !shippingMethod) {
        return 0;
    }

    // Implementation here
    return calculatedCost;
}

// Avoid anonymous functions for better debugging
// Good
var handleAddToCart = function(productId, quantity) {
    // Implementation
};

// Avoid
$('.add-to-cart').click(function() {
    // Anonymous function - harder to debug
});
```

4.3 Error Handling

```
// Always handle errors appropriately
function processPayment(paymentData) {
    try {
        // Payment processing logic
        var result = paymentService.processPayment(paymentData);
        return result;
    } catch (error) {
        // Log the error with context
        Logger.error('Payment processing failed: {0}', error.message);

        // Return appropriate error response
        return {
            error: true,
            message: 'Payment processing failed'
        };
    }
}

// Use specific error types when possible
try {
    // Code that might fail
} catch (e) {
    if (e instanceof ReferenceError) {
        // Handle reference errors
    } else if (e instanceof TypeError) {
        // Handle type errors
    } else {
        // Handle general errors
    }
}
```

4.4 Module Usage

```
// Use CommonJS module pattern
'use strict';

// Import modules at the top
var ProductMgr = require('dw/catalog/ProductMgr');
var URLUtils = require('dw/web/URLUtils');
var Logger = require('dw/system/Logger').getLogger('custom', 'product');

// Module-specific functionality
function getProductById(productId) {
    try {
        return ProductMgr.getProduct(productId);
    } catch (error) {
        Logger.error('Error retrieving product: {0}', error.message);
        return null;
    }
}

// Export public functions
```

```
module.exports = {
    getProductById: getProductById
};
```

5. ISML Template Standards

5.1 Template Structure

```
&lt;iscontent type="text/html" charset="UTF-8" compact="true" /&gt;

&lt;iscomment&gt;
    Template: product-details.isml
    Description: Displays detailed product information
    Requirements: Product object in viewData
    Author: Development Team
    Date: 2025-01-01
&lt;/iscomment&gt;
```

5.2 Security Best Practices

```
<h1>&lt;isprint value="${pdict.product.name}" encoding="htmlcontent" /&gt;</h1>


&lt;iscontent type="text/html" charset="UTF-8" compact="true" /&gt;


&lt;isif condition="${!empty(pdict.product.description)}"&gt;
    <div>
        &lt;isprint value="${pdict.product.description}" encoding="htmlcontent" /&gt;
    </div>
&lt;/isif&gt;
```

5.3 Template Organization

```
&lt;isinclude template="/components/breadcrumbs" /&gt;


&lt;ismodule template="/components/product/productTile"
          name="productTile"
          attribute="product" /&gt;


&lt;isif condition="${!empty(pdict.products)}"&gt;
    &lt;isloop items="${pdict.products}" var="product" status="loopStatus"&gt;
```

```
    &lt;/isloop&gt;
&lt;iselse&gt;
    <div>
        ${Resource.msg('product.search.noproducts', 'product', null)}
    </div>
&lt;/isif&gt;
```

5.4 Asset Management

```
&lt;isscript&gt;
    var assets = require('*/cartridge/scripts/assets');
    assets.addCss('/css/product-detail.css');
    assets.addJs('/js/product-detail.js');
&lt;/isscript&gt;
```

6. Controller Development Guidelines

6.1 Controller Structure

```
'use strict';

var server = require('server');
var Logger = require('dw/system/Logger').getLogger('custom', 'product');

// Import required modules
var ProductMgr = require('dw/catalog/ProductMgr');
var URLUtils = require('dw/web/URLUtils');

/**
 * Product controller - handles product-related routes
 */

// Route: Product-Show
server.get('Show', function (req, res, next) {
    var productId = req.querystring.pid;

    if (!productId) {
        Logger.error('Product ID is required');
        res.redirect(URLUtils.url('Home-Show'));
        return next();
    }

    try {
        var product = ProductMgr.getProduct(productId);

        if (!product) {
            Logger.error('Product not found: {0}', productId);
            res.setStatusCode(404);
            res.render('error/notfound');
            return next();
```

```
        }

        res.render('product/productDetails', {
            product: product
        });

    } catch (error) {
        Logger.error('Error displaying product: {0}', error.message);
        res.redirect(URLUtils.url('Home-Show'));
    }

    next();
});

module.exports = server.exports();
```

6.2 Middleware Pattern

```
// Use middleware for reusable functionality
var csrfProtection = require('*/cartridge/scripts/middleware/csrf');
var userLoggedIn = require('*/cartridge/scripts/middleware/userLoggedIn');

server.post('AddToCart',
    csrfProtection.validateRequest,
    function (req, res, next) {
        // Add to cart logic
        next();
    }
);

server.get('Account-Show',
    userLoggedIn.validateLoggedIn,
    function (req, res, next) {
        // Account display logic
        next();
    }
);
```

6.3 ViewData Management

```
// Use setViewData for extending controller data
server.append('Show', function (req, res, next) {
    var viewData = res.getViewData();

    // Add additional data
    viewData.customRecommendations = getCustomRecommendations(viewData.product);
    viewData.relatedProducts = getRelatedProducts(viewData.product);

    res.setViewData(viewData);
    next();
});
```

7. Model and ViewData Standards

7.1 Model Structure

```
'use strict';

/**
 * Product Model
 * @param {dw.catalog.Product} product - DW Product object
 * @constructor
 */
function ProductModel(product) {
    if (!product) {
        throw new Error('Product is required');
    }

    this.id = product.ID;
    this.name = product.name;
    this.description = product.longDescription ? product.longDescription.markup : '';
    this.price = this.getPrice(product);
    this.images = this.getImages(product);
    this.availability = this.getAvailability(product);
}

/**
 * Get product price information
 * @param {dw.catalog.Product} product - DW Product object
 * @returns {Object} Price information
 */
ProductModel.prototype.getPrice = function (product) {
    var priceModel = product.getPriceModel();

    return {
        sales: priceModel.getPrice().value,
        list: priceModel.getPriceBookPrice('list-prices').value,
        currency: priceModel.getPrice().currencyCode
    };
};

/**
 * Get product images
 * @param {dw.catalog.Product} product - DW Product object
 * @returns {Array} Array of image objects
 */
ProductModel.prototype.getImages = function (product) {
    var images = [];
    var productImages = product.getImages('large');

    for (var i = 0; i &lt; productImages.length; i++) {
        images.push({
            alt: productImages[i].alt,
            url: productImages[i].getURL(),
            title: productImages[i].title
        });
    }
```

```
    return images;
};

module.exports = ProductModel;
```

7.2 ViewData Standards

```
// Controller usage of models
var ProductModel = require('*/cartridge/models/product');

server.get('Show', function (req, res, next) {
    var product = ProductMgr.getProduct(req.querystring.pid);
    var productModel = new ProductModel(product);

    res.render('product/productDetails', {
        product: productModel,
        breadcrumbs: getBreadcrumbs(product),
        pageContext: {
            title: product.name,
            type: 'product'
        }
    });

    next();
});
```

8. CSS and Client-side Asset Management

8.1 CSS Standards

```
// Use SCSS with organized structure
// Variables
$primary-color: #007db8;
$secondary-color: #6c757d;
$font-family-base: 'Helvetica Neue', Helvetica, Arial, sans-serif;

// Mixins
@mixin button-style($bg-color, $text-color: #fff) {
    background-color: $bg-color;
    color: $text-color;
    padding: 0.75rem 1.5rem;
    border: none;
    border-radius: 0.25rem;
    cursor: pointer;

    &amp;:hover {
        background-color: darken($bg-color, 10%);
    }
}
```

```
// Component styles
.product-tile {
    display: flex;
    flex-direction: column;
    border: 1px solid #dee2e6;
    border-radius: 0.25rem;

    &amp;__image {
        position: relative;
        width: 100%;
        padding-bottom: 100%; // 1:1 aspect ratio

        img {
            position: absolute;
            top: 0;
            left: 0;
            width: 100%;
            height: 100%;
            object-fit: cover;
        }
    }

    &amp;__content {
        padding: 1rem;
        flex-grow: 1;
    }

    &amp;__title {
        font-size: 1.125rem;
        font-weight: 600;
        margin-bottom: 0.5rem;
    }

    &amp;__price {
        font-size: 1.25rem;
        font-weight: 700;
        color: $primary-color;
    }
}
```

8.2 JavaScript Client-side Standards

```
'use strict';

// Module pattern for client-side JavaScript
var ProductDetail = (function () {
    var config = {
        selectors: {
            addToCartBtn: '.add-to-cart-btn',
            quantitySelector: '.quantity-select',
            variantSelector: '.variant-selector'
        },
        urls: {
            addToCart: $('.add-to-cart-form').attr('action')
        }
```

```
};

/**
 * Handle add to cart functionality
 */
function handleAddToCart() {
    $(config.selectors.addToCartBtn).on('click', function (e) {
        e.preventDefault();

        var form = $(this).closest('form');
        var formData = form.serialize();

        $.ajax({
            url: config.urls.addToCart,
            method: 'POST',
            data: formData,
            success: function (response) {
                if (response.success) {
                    // Handle success
                    showSuccessMessage(response.message);
                } else {
                    // Handle error
                    showErrorMessage(response.error);
                }
            },
            error: function (xhr, status, error) {
                console.error('Add to cart failed:', error);
                showErrorMessage('Unable to add item to cart');
            }
        });
    });
}

/**
 * Show success message to user
 * @param {string} message - Success message
 */
function showSuccessMessage(message) {
    // Implementation for showing success message
    console.log('Success:', message);
}

/**
 * Show error message to user
 * @param {string} message - Error message
 */
function showErrorMessage(message) {
    // Implementation for showing error message
    console.error('Error:', message);
}

/**
 * Initialize the module
 */
function init() {
    handleAddToCart();
```

```
    }

    // Public API
    return {
        init: init
    };
})();

// Initialize when document is ready
$(document).ready(function () {
    ProductDetail.init();
});
```

9. Security Best Practices

9.1 Input Validation and Sanitization

```
// Server-side validation
function validateProductData(productData) {
    var errors = [];

    // Validate required fields
    if (!productData.id || typeof productData.id !== 'string') {
        errors.push('Product ID is required and must be a string');
    }

    // Sanitize inputs
    if (productData.description) {
        productData.description = sanitizeHtml(productData.description);
    }

    // Validate against business rules
    if (productData.quantity &amp;amp;&amp;amp; (productData.quantity &lt; 1 || productData.quant
        errors.push('Quantity must be between 1 and 99');
    }

    return {
        isValid: errors.length === 0,
        errors: errors,
        data: productData
    };
}

// HTML sanitization helper
function sanitizeHtml(input) {
    if (typeof input !== 'string') return '';

    return input
        .replace(/&lt;/g, '&lt;')
        .replace(/&gt;/g, '&gt;')
        .replace(/"/g, '&quot;')
        .replace(/'/g, '&#x27;')
```

```
        .replace(/\//g, '&#x2F;');
}
```

9.2 CSRF Protection

```
// CSRF middleware
var CSRFProtection = require('dw/web/CSRFProtection');

server.post('UpdateProfile',
    function (req, res, next) {
        // Validate CSRF token
        if (!CSRFProtection.validateRequest()) {
            res.setStatusCode(403);
            res.json({
                error: true,
                message: 'Invalid request'
            });
            return next();
        }

        // Process request
        next();
    }
);
```

9.3 Secure Headers Configuration

```
// HTTP headers configuration
// File: cartridge/config/httpHeadersConf.json
[
    {
        "id": "X-Frame-Options",
        "value": "SAMEORIGIN"
    },
    {
        "id": "X-Content-Type-Options",
        "value": "nosniff"
    },
    {
        "id": "X-XSS-Protection",
        "value": "1; mode=block"
    },
    {
        "id": "Content-Security-Policy",
        "value": "default-src 'self'; script-src 'self' 'unsafe-inline' 'unsafe-eval'"
    }
]
```

10. Error Handling and Logging

10.1 Logging Standards

```
'use strict';

var Logger = require('dw/system/Logger');

// Create category-specific loggers
var productLogger = Logger.getLogger('custom', 'product');
var paymentLogger = Logger.getLogger('custom', 'payment');
var integrationLogger = Logger.getLogger('custom', 'integration');

// Log levels: ERROR, WARN, INFO, DEBUG
function processOrder(orderData) {
    try {
        productLogger.info('Processing order: {0}', orderData.orderNumber);

        // Processing logic here

        productLogger.info('Order processed successfully: {0}', orderData.orderNumber);

    } catch (error) {
        productLogger.error('Error processing order {0}: {1}',
                            orderData.orderNumber, error.message);

        // Don't expose internal errors to client
        throw new Error('Order processing failed');
    }
}

// Structured logging for better analysis
function logPaymentTransaction(transaction) {
    paymentLogger.info('Payment transaction: {0}', JSON.stringify({
        transactionId: transaction.id,
        amount: transaction.amount.value,
        currency: transaction.amount.currencyCode,
        status: transaction.status,
        timestamp: new Date().toISOString()
    }));
}
```

10.2 Error Response Patterns

```
// Consistent error response format
function createErrorResponse(errorType, message, details) {
    return {
        error: true,
        errorType: errorType,
        message: message,
        details: details || null,
        timestamp: new Date().toISOString()
    };
```

```
}

// Usage in controllers
server.post('AddToCart', function (req, res, next) {
    try {
        var result = addProductToCart(req.form);

        if (result.success) {
            res.json({
                success: true,
                cartTotal: result.cartTotal
            });
        } else {
            res.json(createErrorResponse('VALIDATION_ERROR',
                                         result.message,
                                         result.validationErrors));
        }

    } catch (error) {
        Logger.error('Add to cart error: {0}', error.message);

        res.setStatusCode(500);
        res.json(createErrorResponse('INTERNAL_ERROR',
                                     'Unable to add item to cart'));
    }

    next();
});
```

11. Testing Standards

11.1 Unit Testing

```
// Test file: test/unit/models/product.js
'use strict';

var assert = require('chai').assert;
var proxyquire = require('proxyquire').noCallThru();

describe('ProductModel', function () {
    var ProductModel;
    var mockProduct;

    beforeEach(function () {
        mockProduct = {
            ID: 'test-product',
            name: 'Test Product',
            longDescription: {
                markup: 'Test description'
            },
            getPriceModel: function () {
                return {
                    getPrice: function () {
```

```
                        return { value: 29.99, currencyCode: 'USD' };
                    },
                    getPriceBookPrice: function () {
                        return { value: 39.99 };
                    }
                };
            }
        };

        ProductModel = proxyquire('../../../cartridge/models/product', {});
    });

    describe('#constructor', function () {
        it('should create product model with valid product', function () {
            var model = new ProductModel(mockProduct);

            assert.equal(model.id, 'test-product');
            assert.equal(model.name, 'Test Product');
            assert.equal(model.description, 'Test description');
        });

        it('should throw error with invalid product', function () {
            assert.throws(function () {
                new ProductModel(null);
            }, 'Product is required');
        });
    });

    describe('#getPrice', function () {
        it('should return correct price information', function () {
            var model = new ProductModel(mockProduct);

            assert.equal(model.price.sales, 29.99);
            assert.equal(model.price.list, 39.99);
            assert.equal(model.price.currency, 'USD');
        });
    });
});
```

11.2 Integration Testing

```
// Test file: test/integration/controllers/Product.js
'use strict';

var assert = require('chai').assert;
var request = require('request-promise');
var config = require('../config');

describe('Product Controller', function () {

    describe('Product-Show route', function () {
        it('should return product details for valid product ID', function () {
            var options = {
                url: config.baseUrl + '/Product-Show?pid=test-product',
                method: 'GET',
```

```
                resolveWithFullResponse: true
            };

            return request(options)
                .then(function (response) {
                    assert.equal(response.statusCode, 200);
                    assert.include(response.body, 'test-product');
                });
        });

        it('should return 404 for invalid product ID', function () {
            var options = {
                url: config.baseUrl + '/Product-Show?pid=invalid-product',
                method: 'GET',
                resolveWithFullResponse: true,
                simple: false
            };

            return request(options)
                .then(function (response) {
                    assert.equal(response.statusCode, 404);
                });
        });
    });
});
```

11.3 Test Configuration

```
// package.json test scripts
{
    "scripts": {
        "test": "npm run test:unit &amp;amp;&amp;amp; npm run test:integration",
        "test:unit": "mocha test/unit/**/*.js --recursive",
        "test:integration": "mocha test/integration/**/*.js --recursive",
        "test:functional": "webdriver.io",
        "cover": "nyc npm run test:unit",
        "lint": "eslint cartridge/"
    }
}
```

12. Documentation Requirements

12.1 Code Comments

```
/**
 * Calculate shipping cost for given parameters
 *
 * @param {Object} shippingData - Shipping calculation parameters
 * @param {string} shippingData.method - Shipping method ID
 * @param {Object} shippingData.address - Shipping address
 * @param {Array} shippingData.items - Array of cart items
```

```
 * @param {number} shippingData.subtotal - Cart subtotal
 *
 * @returns {Object} Shipping calculation result
 * @returns {boolean} returns.success - Whether calculation succeeded
 * @returns {number} returns.cost - Calculated shipping cost
 * @returns {string} returns.currency - Currency code
 * @returns {string} returns.error - Error message if calculation failed
 *
 * @throws {Error} When required parameters are missing
 *
 * @example
 * var result = calculateShipping({
 *     method: 'standard',
 *     address: customerAddress,
 *     items: cart.items,
 *     subtotal: 150.00
 * });
 */
function calculateShipping(shippingData) {
    // Implementation here
}
```

12.2 ISML Template Documentation

```
&lt;iscomment&gt;
    Template: product/productTile.isml

    Description:
        Renders a product tile component for use in product listing pages,
        search results, and recommendation sections.

    Parameters:
        @param {Object} product - Product model object
        @param {boolean} showPrice - Whether to display price information
        @param {string} tileSize - Size variant: 'small', 'medium', 'large'
        @param {boolean} showRating - Whether to display product rating

    Dependencies:
        - ProductModel
        - PriceHelper
        - URLUtils

    Example Usage:
        &lt;isinclude template="product/productTile" sf:product="${product}"
                   sf:showPrice="${true}" sf:tileSize="medium" /&gt;

    Author: Development Team
    Created: 2025-01-01
    Last Modified: 2025-01-01
&lt;/iscomment&gt;
```

12.3 README Documentation

```
# Custom SFCC Cartridge

## Overview
This cartridge provides enhanced product functionality for the SFCC storefront.

## Installation
1. Upload cartridge to SFCC instance
2. Add to cartridge path in Business Manager
3. Import metadata objects
4. Configure custom preferences

## Configuration
### Site Preferences
- `enableCustomFeatures`: Enable/disable custom functionality
- `customAPIEndpoint`: External API endpoint URL

### Environment Variables
- `CUSTOM_API_KEY`: API key for external service integration

## Features
- Enhanced product recommendations
- Custom pricing logic
- Advanced search functionality

## API Documentation
See `/docs/api.md` for detailed API documentation.

## Testing
Run tests with: `npm test`

## Deployment
Follow standard SFCC deployment procedures.
```

13. Performance Optimization

13.1 Database Query Optimization

```
// Avoid queries in loops
// Bad
products.forEach(function(product) {
    var relatedProducts = ProductMgr.queryAllSiteProducts('category = {0}',
                                                           product.getPrimaryCategory().getI
});

// Good - batch queries
var categoryIds = products.map(function(product) {
    return product.getPrimaryCategory().getID();
});
var allRelatedProducts = ProductMgr.queryAllSiteProducts('category IN {0}', categoryIds);
```

13.2 Cache Utilization

```
var CacheMgr = require('dw/system/CacheMgr');

function getProductRecommendations(productId) {
    var cache = CacheMgr.getCache('ProductRecommendations');
    var cacheKey = 'recommendations_' + productId;

    // Try to get from cache first
    var recommendations = cache.get(cacheKey);

    if (!recommendations) {
        // Generate recommendations if not cached
        recommendations = generateRecommendations(productId);

        // Cache for 1 hour
        cache.put(cacheKey, recommendations, 3600);
    }

    return recommendations;
}
```

13.3 Template Performance

```


&lt;iscache type="relative" hour="24" /&gt;


&lt;isloop items="${pdict.products}" var="product" status="loopState"&gt;
    &lt;isif condition="${loopState.first}"&gt;
        <div>
    &lt;/isif&gt;



    &lt;isif condition="${loopState.last}"&gt;
        </div>
    &lt;/isif&gt;
&lt;/isloop&gt;



&lt;isif condition="${product.getAvailabilityModel().isOrderable() &amp;&amp; product.get


&lt;isif condition="${product.isAvailableAndPriced}"&gt;
```

14. Code Review and Quality Assurance

14.1 Code Review Checklist

Functionality

- [ ] Code meets requirements and specifications
- [ ] Edge cases are handled appropriately
- [ ] Error handling is comprehensive
- [ ] No hardcoded values where configuration should be used

Security

- [ ] Input validation is implemented
- [ ] Output encoding prevents XSS
- [ ] CSRF protection is used where needed
- [ ] Sensitive data is not logged

Performance

- [ ] No unnecessary database queries
- [ ] Caching is used appropriately
- [ ] Resource usage is optimized

Code Quality

- [ ] Follows naming conventions
- [ ] Code is readable and well-documented
- [ ] No code duplication
- [ ] Consistent formatting and style

Testing

- [ ] Unit tests are included
- [ ] Test coverage is adequate
- [ ] Integration tests cover key workflows

14.2 ESLint Configuration

```
{
    "extends": ["eslint:recommended"],
    "env": {
        "browser": true,
        "node": true,
        "es6": false
    },
    "globals": {
```

```
        "empty": "readonly",
        "session": "readonly",
        "request": "readonly",
        "response": "readonly",
        "customer": "readonly"
    },
    "rules": {
        "indent": ["error", 4],
        "quotes": ["error", "single"],
        "semi": ["error", "always"],
        "no-unused-vars": "warn",
        "no-console": "warn",
        "brace-style": ["error", "1tbs"],
        "comma-dangle": ["error", "never"],
        "max-len": ["warn", {"code": 120}]
    }
}
```

15. Collaboration and Deployment Practices

15.1 Version Control Standards

```
# Branch naming conventions
feature/product-recommendations
bugfix/cart-calculation-error
hotfix/security-vulnerability
release/v2.1.0

# Commit message format
feat: add product recommendation engine
fix: resolve cart total calculation issue
docs: update API documentation
test: add unit tests for ProductModel
refactor: optimize database queries
```

15.2 Deployment Process

```
# Example CI/CD pipeline configuration
stages:
  - lint
  - test
  - build
  - deploy-staging
  - integration-tests
  - deploy-production

lint:
  script:
    - npm run lint
    - npm run stylelint
```

```
test:
  script:
    - npm run test:unit
    - npm run test:coverage

build:
  script:
    - npm run build
    - npm run compile:js
    - npm run compile:scss

deploy-staging:
  script:
    - sfcc-deploy staging
  environment: staging
  only:
    - develop

deploy-production:
  script:
    - sfcc-deploy production
  environment: production
  only:
    - main
  when: manual
```

15.3 Environment Management

```
// Environment-specific configuration
// File: cartridge/config/environment.js

var environment = {
    development: {
        apiEndpoint: 'https://dev-api.example.com',
        enableDebugLogging: true,
        cacheTimeout: 300
    },
    staging: {
        apiEndpoint: 'https://staging-api.example.com',
        enableDebugLogging: false,
        cacheTimeout: 1800
    },
    production: {
        apiEndpoint: 'https://api.example.com',
        enableDebugLogging: false,
        cacheTimeout: 3600
    }
};

var currentEnv = System.getProperty('environment') || 'development';

module.exports = environment[currentEnv];
```

Conclusion

This coding style guide provides a comprehensive framework for developing high-quality SFCC B2C cartridges. By following these standards, development teams can create maintainable, secure, and performant code that scales with business requirements.

Key Takeaways

1. **Consistency**: Follow naming conventions and coding standards across all team members
2. **Security**: Always validate inputs, encode outputs, and follow security best practices
3. **Performance**: Optimize database queries, use caching, and minimize resource usage
4. **Testing**: Include comprehensive unit, integration, and functional tests
5. **Documentation**: Document code thoroughly for future maintainability
6. **Collaboration**: Use version control best practices and standardized deployment processes

Continuous Improvement

This guide should be reviewed and updated regularly to incorporate new best practices, security updates, and platform changes. Team feedback and lessons learned from projects should be incorporated to keep the guide relevant and useful.

Document Version: 1.0
Last Updated: January 2025
Next Review Date: July 2025

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50] [51] [52] [53] [54] [55] [56] [57] [58] [59]

⁂

1. https://www.ranosys.com/blog/insights/how-sfcc-cartridges-help-tailor-modern-digital-experiences/
2. https://developer.salesforce.com/docs/commerce/b2c-commerce/guide/b2c-cartridges.html
3. https://developer.salesforce.com/docs/commerce/sfra/guide/b2c-sfra-standards-compliance.html
4. https://developer.salesforce.com/docs/commerce/b2c-commerce/guide/b2c-dev-best-practices.html
5. https://developer.salesforce.com/docs/commerce/b2c-commerce/guide/b2c-site-performance.html
6. https://www.rhino-inquisitor.com/how-to-load-client-side-javascript-and-css-in-sfra/
7. https://felkamcommerce.com/mastering-isml-templates-in-sfcc/
8. https://help.salesforce.com/s/articleView?id=mktg.mc_as_filename_patterns_reference.htm&language=en_US&type=5
9. https://developer.salesforce.com/docs/commerce/sfra/guide/b2c-sfra-features-and-comps.html
10. https://sfcclearning.com/infocenter/content/b2c_commerce/topics/site_development/b2c_content_best_practices.php
11. https://trailhead.salesforce.com/content/learn/modules/success-cloud-coding-conventions/choose-naming-conventions-sc

12. https://sfcclearning.com/infocenter/content/b2c_commerce/topics/site_performance/b2c_performance_and_stability_coding_standards.php
13. https://developer.salesforce.com/docs/commerce/sfra/guide/b2c-sfra-modules.html
14. https://www.linkedin.com/posts/maxime-rebibo-sfcc_sfcc-salesforcecommercecloud-sfcctips-activity-7206180859173847042-Mqgs
15. https://help.salesforce.com/s/articleView?id=data.c360_a_data_lake_object_naming.htm&language=en_US&type=5
16. https://salesforcecommercecloud.github.io/b2c-dev-doc/docs/current/sfrajsdoc/js/server/modules_server_response.js.html
17. https://www.sfcclearning.com/blog/isml-template-best-practices/
18. https://help.salesforce.com/s/articleView?id=sf.dato_harmonize_center_naming.htm&language=en_US&type=5
19. https://developer.salesforce.com/docs/commerce/b2c-commerce/guide/b2c-working-with-templates.html
20. https://www.salesforceben.com/salesforce-naming-conventions-how-to-enforce-them/
21. https://sfcclearning.com/infocenter/content/b2c_commerce/topics/isml/b2c_isml.php
22. https://gtcsys.com/faq/what-are-the-best-practices-for-error-handling-and-graceful-error-recovery-in-web-application-development/
23. https://developer.salesforce.com/docs/commerce/sfra/guide/b2c-customizing-sfra.html
24. https://www.walkme.com/blog/salesforce-commerce-cloud/
25. https://cyntexa.com/blog/customer-data-privacy-with-commerce-cloud/
26. https://sfdclesson.com/2023/11/20/apex-exception-handling-best-practices-for-developers/
27. https://www.leapwork.com/blog/what-is-salesforce-testing
28. https://blog.blueinfy.com/2024/08/performing-security-code-review-for.html
29. https://docs.workato.com/recipes/best-practices-error-handling.html
30. https://trailhead.salesforce.com/content/learn/modules/cc-project-mgmt/cc-test-implementation
31. https://tradecentric.com/blog/salesforce-commerce-cloud-security/
32. https://devopslaunchpad.com/blog/advanced-apex-error-handling/
33. https://developer.salesforce.com/docs/commerce/sfra/guide/b2c-testing-sfra.html
34. https://www.sunriseintegration.com/blogs/learn/developing-cartridges-for-salesforce-commerce-cloud
35. https://help.salesforce.com/s/articleView?id=cc.b2c_security_best_practices_for_developers.htm&language=en_US&type=5
36. https://developer.salesforce.com/blogs/2020/08/error-handling-best-practices-for-lightning-web-components
37. https://pivotal.digital/integrations/p/salesforce-commerce-cloud
38. https://help.salesforce.com/s/articleView?id=cc.b2c_ab_test_resources.htm&language=en_US&type=5
39. https://help.powerreviews.com/hc/en-us/articles/17399942921627-Salesforce-Commerce-Cloud-Cartridge
40. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/langCon_apex_expressions_comments.htm
41. https://moldstud.com/articles/p-what-are-the-best-practices-for-collaborating-with-remote-salesforce-developers

42. https://amplience.com/developers/docs/integrations/commerce/sfcc-cartridge/
43. https://docs.uniform.app/docs/integrations/commerce/salesforce-b2c-commerce
44. https://gearset.com/blog/how-to-improve-collaboration/
45. https://trailhead.salesforce.com/content/learn/modules/success-cloud-coding-conventions/get-started-with-coding-conventions-sc
46. https://help.ordergroove.com/hc/en-us/articles/360057610254-Salesforce-Integration-Guide-Cartridge-Overview
47. https://sfcclearning.com/infocenter/content/b2c_commerce/topics/b2c_security_best_practices/b2c_general_secure_coding_practices.php
48. https://www.codleo.com/blog/salesforce-development-best-practices-2025
49. https://docs.subscribepro.com/technical/release-notes/salesforce-commerce-cloud-cartridge/
50. https://www.youtube.com/watch?v=fCcxBe_h7ow
51. https://sfcclearning.com/infocenter/content/b2c_commerce/topics/versioning/b2c_collaborative_development_and_deployment.php
52. https://cloudinary.com/documentation/salesforce_commerce_cloud_cartridge_integration
53. https://www.scribd.com/document/658585743/CCD-101-Developing-for-B2C-Commerce-Student-Guide-Aug-2018
54. https://trailhead.salesforce.com/content/learn/trails/create-inclusive-collaboration-experiences-during-the-design-process
55. https://www.youtube.com/watch?v=qNK97LFlYLg
56. https://trailhead.salesforce.com/content/learn/modules/b2c-cartridges/b2c-cartridges-explore
57. https://sfcclearning.com/infocenter/content/b2c_commerce/topics/sfra/b2c_building_your_cartridge_stack.php
58. https://www.rhino-inquisitor.com/secure-coding-in-salesforce-b2c-commerce-cloud/
59. https://sfcclearning.com/infocenter/sfrajsdoc/js/server/app_storefront_base_cartridge_scripts_assets.js.php