1. 顧客管理と価格設定

【課題】
顧客別の商品表示や複雑な価格設定など、CS-Cart の標準機能では対応しきれないカスタマイズ

【取り組みと成果】

a）顧客別商品表示システム
- 新たにカスタムテーブル `custom_product_visibility` を設計・実装
- 商品一覧表示ロジックを完全に書き換え、パフォーマンスを維持しつつ柔軟な表示制御を実現
- 管理画面に直感的な設定インターフェースを追加（約 40 時間の開発工数）

b）複雑な価格設定システム
- `custom_price_rules` テーブルを新設し、多次元の価格ルールに対応
- 価格計算エンジンを一から再設計し、R.CODE のアソート販売のような複雑なルールにも対応
- キャッシュシステムを導入し、複雑な計算による速度低下を最小限に抑制（約 60 時間の最適化作業）

c）管理画面の拡張
- ユーザビリティを重視し、複雑な設定を簡単に行える UI を設計
- プレビュー機能を追加し、設定結果をリアルタイムで確認可能に

2. 商品管理

【課題】
季節商品の管理や高度な在庫管理機能が標準では不十分でした。

【取り組みと成果】

a）季節商品管理システム
- 商品テーブルを拡張し、表示期間管理を可能に
- バッチ処理を実装し、深夜に自動で商品の表示/非表示を切り替え

- 特定顧客向けの例外設定機能を追加（約 30 時間の開発工数）

b）在庫管理システムの拡張
- リアルタイム在庫監視システムを実装し、設定した閾値を下回ると自動通知
- 予約在庫、取り寄せ在庫など、複雑な在庫状態に対応
- 在庫データの一括インポート/エクスポート機能を強化（約 50 時間の開発と最適化）

3. 注文処理と配送

【課題】
多様な決済方法への対応と、ユーザー直送モードの実装が必要でした。

【取り組みと成果】

a）複雑な決済システム
- 売掛金管理システムを新規開発し、与信限度額の自動チェック機能を実装
- 先入金システムと連携し、入金確認から出荷までの一連の流れを自動化
- 顧客ごとの決済方法制御を可能に（約 45 時間の開発工数）

b）ユーザー直送モード
- 注文プロセス全体を見直し、直送モードに対応するよう再設計
- 直送時の価格自動調整ロジックを実装し、利益率を維持
- 画面デザインを工夫し、モード間の視覚的区別を明確化（約 55 時間の開発とデザイン作業）

4. キャンペーンと割引

【課題】
より柔軟で効果的なキャンペーン管理と、その効果測定が求められました。

【取り組みと成果】

a）高度なキャンペーン管理システム
- 条件設定エンジンを一から開発し、複雑な条件の組み合わせに対応
- ルールベースのプロモーション適用ロジックを実装し、高速な処理を実現
- 管理画面にビジュアルなルールビルダーを実装（約 70 時間の開発工数）

b）キャンペーンの自動化
- クーロンジョブシステムと連携し、キャンペーンの自動開始/終了を実現
- 管理者への自動通知システムを実装し、キャンペーン状況を常時把握可能に

c）効果測定と分析機能
- 詳細なログ記録システムを実装し、キャンペーンの影響を正確に追跡
- ダッシュボード機能を追加し、キャンペーンの ROI をリアルタイムで可視化（約 40 時間の開発とデータ分析）

5. システム統合と最適化

【課題】
外部システムとの連携と、新機能追加に伴うパフォーマンスの最適化が必要でした。

【取り組みと成果】

a) API の拡張
- RESTful API を拡張し、新規追加したすべての機能にアクセス可能に
- OAuth2.0 による認証システムを実装し、セキュリティを強化（約 35 時間の開発工数）

b）データエクスポート機能
- Freee 用のカスタムエクスポーターを開発し、シームレスなデータ連携を実現
- バッチ処理による自動エクスポート機能を追加し、作業効率を向上

c）パフォーマンス最適化
- データベースクエリを全面的に見直し、インデックスを最適化（約 100 のクエリを改善）
- メモリキャッシュシステムを導入し、頻繁なデータアクセスを高速化
- ページロード時間を平均 30%削減（約 80 時間の最適化作業）

CS-Cart カスタマイズ

1. 顧客管理と価格設定

1.1 顧客別商品表示システム

機能概要:
- 顧客 ID に基づいて表示する商品を制御
- 顧客グループごとの商品表示設定
- 管理画面からの簡単操作

実装詳細:
a) データベース
    - 新テーブル: `custom_product_visibility`
    - 構造:
      * id (INT, PRIMARY KEY)
      * product_id (INT, 外部キー: cscart_products)
      * user_id (INT, 外部キー: cscart_users)

b) バックエンド
    - ファイル: `app/functions/fn.catalog.php`
    - 新規関数: `fn_get_visible_products_for_user($user_id)`
    - 機能: ユーザーID に基づいて表示可能な商品リストを取得

c) フロントエンド
    - ファイル: `app/controllers/frontend/products.php`
    - 修正: 商品一覧取得ロジックに可視性チェックを追加

d) 管理画面
    - 新規ページ: 「顧客別商品表示設定」
    - パス: `admin.php?dispatch=products.customer_visibility`
    - 機能: 商品ごとに表示可能な顧客を設定

使用方法:
1. 管理画面の「商品管理」→「顧客別商品表示設定」に移動

2. 対象商品を選択し、表示を許可する顧客を指定
3. 設定を保存
4. フロントエンドで各顧客に適切な商品のみが表示されることを確認

1.2 複雑な価格設定システム

機能概要:
- 顧客ごとの個別価格設定
- 購入数量に応じた段階的な価格変動
- R.CODE のアソート販売対応

実装詳細:
a) データベース
    - 新テーブル: `custom_price_rules`
    - 構造:
      * id (INT, PRIMARY KEY)
      * user_id (INT, 外部キー: cscart_users)
      * product_id (INT, 外部キー: cscart_products)
      * quantity_from (INT)
      * quantity_to (INT)
      * price_factor (DECIMAL(10,2))

b) バックエンド
    - ファイル: `app/functions/fn.catalog.php`
    - 新規関数: `fn_calculate_custom_price($product_id, $base_price, $quantity, $user_id)`
    - 機能: 顧客、商品、数量に基づいて最終価格を計算

c) フロントエンド
    - ファイル: `app/functions/fn.cart.php`
    - 修正: `fn_calculate_cart_content` 関数内で新しい価格計算ロジックを呼び出し

d) 管理画面
    - 新規ページ: 「カスタム価格ルール設定」
    - パス: `admin.php?dispatch=products.custom_price_rules`
    - 機能: 商品ごとに複雑な価格ルールを設定

使用方法:
1. 管理画面の「商品管理」→「カスタム価格ルール設定」に移動
2. 対象商品を選択し、顧客、数量範囲、価格係数を指定
3. R.CODE アソート販売の場合、特別なルールとして設定
4. 設定を保存
5. フロントエンドで価格が正しく表示・計算されることを確認

2. 商品管理

2.1 季節商品管理システム

機能概要:
- 商品の表示期間設定
- 特定顧客向けの通年表示設定
- 自動的な表示/非表示切り替え

実装詳細:
a) データベース
    - テーブル修正: `cscart_products`
    - 追加フィールド:
      * display_start_date (DATE)
      * display_end_date (DATE)

b) バックエンド
    - ファイル: `app/functions/fn.catalog.php`
    - 新規関数: `fn_check_product_display_date($product_id)`
    - 機能: 現在日付と商品の表示期間を比較

c) クーロンジョブ
    - ファイル: `app/addons/my_changes/func.php`
    - 新規関数: `fn_my_changes_update_product_visibility()`
    - 機能: 毎日深夜に商品の表示状態を更新

d) 管理画面
    - 修正ページ: 商品編集ページ
    - 追加フィールド: 表示開始日、表示終了日、通年表示顧客

使用方法:
1. 商品編集ページで表示開始日と終了日を設定
2. 必要に応じて通年表示する特定顧客を指定
3. 設定を保存
4. フロントエンドで商品の表示/非表示が適切に切り替わることを確認

2.2 在庫管理システムの拡張

機能概要:
- 商品ごとの在庫アラート閾値設定
- リアルタイムの在庫監視と通知
- 予約在庫、取り寄せ在庫の管理

実装詳細:
a) データベース
    - テーブル修正: `cscart_products`
    - 追加フィールド:
      * low_stock_threshold (INT)
      * stock_type (ENUM('regular', 'preorder', 'backorder'))

b) バックエンド
    - ファイル: `app/functions/fn.inventory.php`
    - 新規関数: `fn_check_low_stock_and_notify()`
    - 機能: 在庫レベルをチェックし、閾値を下回った場合に通知

c) 通知システム
    - ファイル: `app/addons/my_changes/func.php`
    - 新規関数: `fn_my_changes_send_stock_notification($product_id, $current_stock)`
    - 機能: 管理者にメール通知を送信

d) 管理画面
    - 修正ページ: 商品編集ページ
    - 追加フィールド: 在庫アラート閾値、在庫タイプ

使用方法:
1. 商品編集ページで在庫アラート閾値と在庫タイプを設定
2. 設定を保存
3. 在庫数が閾値を下回ると自動的に通知が送信されることを確認
4. 予約在庫や取り寄せ在庫の場合、適切な表示と処理が行われることを確認

3. 注文処理と配送

3.1 複雑な決済システム

機能概要:
- 売掛決済オプションの実装
- 先入金システムの導入
- 顧客ごとの決済方法制御

実装詳細:
a) データベース
    - 新テーブル: `custom_credit_limits`
    - 構造:
      * id (INT, PRIMARY KEY)
      * user_id (INT, 外部キー: cscart_users)
      * credit_limit (DECIMAL(12,2))
      * current_balance (DECIMAL(12,2))

b) バックエンド
    - ファイル: `app/payments/invoice.php`
    - 新規関数:
      * `fn_invoice_check_credit_limit($order_info, $processor_data)`
      * `fn_invoice_update_credit_balance($order_info, $processor_data)`

c) フロントエンド
    - ファイル: `app/controllers/frontend/checkout.php`
    - 修正: 決済方法選択時に顧客の利用可能なオプションをフィルタリング

d) 管理画面
    - 新規ページ: 「顧客別決済設定」

- パス: `admin.php?dispatch=customers.payment_methods`
- 機能: 顧客ごとに利用可能な決済方法と与信限度額を設定

使用方法:
1. 管理画面の「顧客管理」→「顧客別決済設定」に移動
2. 対象顧客を選択し、利用可能な決済方法と与信限度額を設定
3. 設定を保存
4. フロントエンドで該当顧客のチェックアウト時に正しい決済オプションが表示されることを確認
5. 売掛決済時に与信限度額のチェックが行われることを確認

3.2 ユーザー直送モード

機能概要:
- エンドユーザーへの直接配送オプション
- 直送時の価格自動調整
- 画面の視覚的フィードバック

実装詳細:
a) データベース
- 新テーブル: `user_direct_shipping`
- 構造:
  * id (INT, PRIMARY KEY)
  * user_id (INT, 外部キー: cscart_users)
  * is_enabled (TINYINT(1))
  * price_increase_factor (DECIMAL(4,2))

b) バックエンド
- ファイル: `app/functions/fn.cart.php`
- 新規関数: `fn_apply_direct_shipping_price_increase(&$cart)`

c) フロントエンド
- ファイル: `js/tygh/checkout.js`
- 新規関数: `fn_change_color_scheme(mode)`

d) テンプレート

ファイル:
`design/themes/[your_theme]/templates/views/checkout/components/steps/shipping.tpl`
    - 修正: 直送モード切替オプションの追加

e) CSS
    - ファイル: `design/themes/[your_theme]/css/addons/direct_shipping/styles.less`
    - 内容: 直送モード時の画面スタイル定義

使用方法:
1. 管理画面で対象顧客のユーザー直送モードを有効化
2. フロントエンドのチェックアウトページで直送モードを選択
3. 画面の配色が変更され、価格が自動調整されることを確認
4. 直送不可の商品が正しく制限されることを確認

4. キャンペーンと割引

4.1 高度なキャンペーン管理システム

機能概要:
- 複雑な条件設定が可能なキャンペーンエンジン
- キャンペーンの自動開始・終了
- 効果測定と分析機能

実装詳細:
a) データベース
    - 新テーブル:
        * `custom_campaigns`
        * `campaign_conditions`
        * `campaign_actions`

b) バックエンド
    - ファイル: `app/functions/fn.promotions.php`
    - 新規関数:
        * `fn_get_active_campaigns()`

    * `fn_check_campaign_conditions($campaign_id, $cart)`
    * `fn_apply_campaign_actions($campaign_id, &$cart)`

c) クーロンジョブ
   - ファイル: `app/addons/my_changes/func.php`
   - 新規関数: `fn_my_changes_manage_campaigns()`

d) 管理画面
   - 新規ページ: 「高度なキャンペーン管理」
   - パス: `admin.php?dispatch=campaigns.manage`
   - 機能: 複雑な条件と行動のルールを視覚的に設定

e) 分析ダッシュボード
   - 新規ページ: 「キャンペーン効果分析」
   - パス: `admin.php?dispatch=campaigns.analytics`
   - 機能: キャンペーンのパフォーマンスをグラフィカルに表示

使用方法:
1. 管理画面の「マーケティング」→「高度なキャンペーン管理」に移動
2. 新規キャンペーンを作成し、条件と行動を設定
3. 開始日と終了日を指定（自動化の場合）
4. キャンペーンを有効化
5. フロントエンドでキャンペーンが正しく適用されることを確認
6. 「キャンペーン効果分析」ページで結果を確認

5. システム統合と最適化

5.1 API の拡張

機能概要:
- 新規カスタム機能への API アクセス
- OAuth2.0 による認証
- Webhook サポート

実装詳細:
a) バックエンド

- ファイル: `app/addons/my_changes/controllers/common/api.php`
- 新規エンドポイント:
  * GET /api/custom_price_rules
  * POST /api/campaigns
  * etc.

b) 認証
- ファイル: `app/addons/my_changes/init.php`
- OAuth2.0 プロバイダーの設定

c) Webhook
- ファイル: `app/addons/my_changes/controllers/common/webhooks.php`
- イベントトリガーの設定

使用方法:
1. 管理画面で API アクセスキーを生成
2. Postman などのツールを使用して API をテスト
3. 外部システムとの連携をセットアップ

5.2 パフォーマンス最適化

機能概要:
- データベースクエリの最適化
- キャッシュシステムの導入
- 非同期処理の実装

実装詳細:
a) データベース
- インデックスの追加
- クエリの書き換え

b) キャッシュ
- ファイル: `app/functions/fn.common.php`
- 新規関数: `fn_get_cached_data($key, $regenerate_callback, $ttl = 3600)`

c) 非同期処理

- 新規アドオン: `background_processing`
- 長時間かかる処理をキューに入れて非同期実行

使用方法:
1. 管理画面の「設定」→「パフォーマンス」で最適化設定を調整
2. キャッシュの有効期限やサイズを設定
3. 非同期処理が必要な重い処理を識別し、バックグラウンド実行に移行