# Incentiva Loyalty Campaign Management System

## Design Document

### Executive Summary

The Incentiva Loyalty Campaign Management System is a comprehensive platform designed to enable companies to create, manage, and execute sophisticated loyalty campaigns. The system leverages AI-powered recommendation engines and natural language processing to translate business requirements into executable loyalty programs using the TLP (Trusted Loyalty Platform) APIs.

**Key Features:** - AI-powered loyalty campaign recommendations - Natural language rules engine for campaign definition - Integration with TLP APIs for point management and offer creation - Multi-language support (English, Portuguese, Spanish) - Dynamic campaign execution with real-time point allocation

### System Architecture

#### Technology Stack

* **Frontend**: React with TypeScript
* **Backend**: Node.js/TypeScript middleware
* **Database**: PostgreSQL
* **AI Integration**: Claude 3.5 Sonnet (recommended for complex reasoning and code generation)
* **Loyalty Platform**: TLP APIs

#### System Components

1. **Campaign Design Interface**
   * Natural language campaign definition
   * AI-powered recommendations
   * Schema ingestion and understanding
2. **Rules Engine**
   * Natural language to code translation
   * Database schema analysis
   * TLP API integration code generation
3. **Campaign Execution Engine**
   * Real-time point allocation
   * Goal tracking and achievement
   * Offer management
4. **TLP Integration Layer**
   * Point type management
   * Offer creation and management
   * Transaction processing

### AI Model Recommendation

**Recommended AI Model: Claude 3.5 Sonnet**

**Rationale:** - Superior reasoning capabilities for complex business logic translation - Excellent code generation with context understanding - Strong performance with structured data and schema analysis - Multi-language support for Portuguese and Spanish - Reliable API integration and documentation generation

### Core Workflow

#### 1. Campaign Design Phase

**Schema Ingestion:** - AI analyzes the sales order application/database schema - Understands relationships between tables (sales order, order detail, salesperson, product, etc.) - Creates a knowledge base for rule generation - Provides feedback on schema understanding success rate

**Natural Language Campaign Definition:** Users specify campaign parameters in natural language:

Goals:  
- Individual goal: 50,000 Brazilian Reals in Premium Line sales  
- Regional goal: 500,000 Brazilian Reals in Premium Line sales  
  
Eligibility Criteria:  
- Users must be actively employed until the last day of the campaign  
- Only products from premium line are eligible  
- Cancelled or returned sales are not eligible  
  
Prizes:  
- 50,000 points: Vacation to Cabo  
- 100,000 points: Concert tickets

#### 2. Rules Engine Processing

**Schema Analysis Output:**

Required Database Access:  
- Tables: orders, order\_line\_items, salespersons, products, product\_lines  
- Fields: order\_id, order\_date, salesperson\_id, product\_id, product\_line,   
 order\_status, order\_amount, region\_id  
- Relationships: orders -> salespersons, orders -> order\_line\_items -> products

**Generated Code Structure:**

interface CampaignRules {  
 individualGoal: number;  
 regionalGoal: number;  
 startDate: Date;  
 endDate: Date;  
 eligibleProductLines: string[];  
 excludedOrderStatuses: string[];  
}  
  
class PremiumLineCampaign {  
 async calculateIndividualPoints(salespersonId: string): Promise<number> {  
 // AI-generated code for individual goal calculation  
 }  
   
 async calculateRegionalPoints(regionId: string): Promise<number> {  
 // AI-generated code for regional goal calculation  
 }  
   
 async allocatePoints(salespersonId: string, points: number): Promise<void> {  
 // TLP API integration code  
 }  
}

#### 3. TLP Integration

**Point Type Creation:**

// Using TLP API: POST /points  
const pointType = {  
 name: "Premium Line Campaign Points",  
 description: "Points earned through Premium Line sales campaign",  
 rank: 1,  
 enabled: true,  
 options: {  
 showZeroPointBalance: true  
 }  
};

**Offer Creation:**

// Individual Goal Offer  
const individualOffer = {  
 offerType: "accrual",  
 offerSubtype: "dynamic",  
 name: "Premium Line Individual Goal",  
 description: "Earn points for individual Premium Line sales goals",  
 points: 50000,  
 pointType: "point.premium\_campaign",  
 minimumSpend: 50000,  
 enabled: true,  
 online: false,  
 location: true  
};  
  
// Regional Goal Offer  
const regionalOffer = {  
 offerType: "accrual",   
 offerSubtype: "dynamic",  
 name: "Premium Line Regional Goal",  
 description: "Earn points for regional Premium Line sales goals",  
 points: 100000,  
 pointType: "point.premium\_campaign",  
 minimumSpend: 500000,  
 enabled: true,  
 online: false,  
 location: true  
};  
  
// Redemption Offers with Auto-Generated Graphics  
const vacationOffer = {  
 offerType: "redemption",  
 offerSubtype: "product",  
 name: "Vacation to Cabo",  
 description: "Redeem 50,000 points for a vacation to Cabo",  
 points: 50000,  
 pointType: "point.premium\_campaign",  
 enabled: true,  
 imageUrl: "https://generated-graphics.incentiva.com/vacation-cabo.png" // Auto-generated  
};  
  
const concertOffer = {  
 offerType: "redemption",  
 offerSubtype: "product",   
 name: "Concert Tickets",  
 description: "Redeem 100,000 points for concert tickets",  
 points: 100000,  
 pointType: "point.premium\_campaign",  
 enabled: true,  
 imageUrl: "https://generated-graphics.incentiva.com/concert-tickets.png" // Auto-generated  
};

### Database Schema

#### Core Tables

**campaigns**

CREATE TABLE campaigns (  
 id UUID PRIMARY KEY,  
 name VARCHAR(255) NOT NULL,  
 description TEXT,  
 start\_date DATE NOT NULL,  
 end\_date DATE NOT NULL,  
 status VARCHAR(50) DEFAULT 'draft',  
 tlp\_api\_key VARCHAR(255),  
 tlp\_endpoint\_url VARCHAR(500),  
 backend\_connection\_config JSONB,  
 created\_by UUID REFERENCES users(id),  
 created\_at TIMESTAMP DEFAULT NOW(),  
 updated\_at TIMESTAMP DEFAULT NOW()  
);

**campaign\_rules**

CREATE TABLE campaign\_rules (  
 id UUID PRIMARY KEY,  
 campaign\_id UUID REFERENCES campaigns(id),  
 rule\_type VARCHAR(50) NOT NULL, -- 'goal', 'eligibility', 'prize'  
 rule\_definition JSONB NOT NULL,  
 generated\_code TEXT,  
 schema\_understanding\_score DECIMAL(3,2), -- 0.00 to 1.00  
 schema\_feedback TEXT,  
 created\_at TIMESTAMP DEFAULT NOW()  
);

**campaign\_executions**

CREATE TABLE campaign\_executions (  
 id UUID PRIMARY KEY,  
 campaign\_id UUID REFERENCES campaigns(id),  
 salesperson\_id VARCHAR(255),  
 region\_id VARCHAR(255),  
 points\_allocated INTEGER DEFAULT 0,  
 goal\_achieved BOOLEAN DEFAULT FALSE,  
 execution\_date TIMESTAMP DEFAULT NOW()  
);

**campaign\_schemas**

CREATE TABLE campaign\_schemas (  
 id UUID PRIMARY KEY,  
 campaign\_id UUID REFERENCES campaigns(id),  
 schema\_definition JSONB NOT NULL,  
 understanding\_score DECIMAL(3,2),  
 feedback\_text TEXT,  
 uploaded\_at TIMESTAMP DEFAULT NOW()  
);

**users**

CREATE TABLE users (  
 id UUID PRIMARY KEY,  
 email VARCHAR(255) UNIQUE NOT NULL,  
 password\_hash VARCHAR(255),  
 first\_name VARCHAR(100),  
 last\_name VARCHAR(100),  
 role VARCHAR(50) NOT NULL, -- 'admin', 'participant'  
 oauth\_provider VARCHAR(50), -- 'google', 'microsoft', etc.  
 oauth\_id VARCHAR(255),  
 created\_at TIMESTAMP DEFAULT NOW(),  
 updated\_at TIMESTAMP DEFAULT NOW()  
);

**user\_campaigns**

CREATE TABLE user\_campaigns (  
 id UUID PRIMARY KEY,  
 user\_id UUID REFERENCES users(id),  
 campaign\_id UUID REFERENCES campaigns(id),  
 participant\_id VARCHAR(255), -- TLP member ID  
 current\_points INTEGER DEFAULT 0,  
 goal\_progress DECIMAL(5,2) DEFAULT 0.00, -- Percentage  
 created\_at TIMESTAMP DEFAULT NOW(),  
 updated\_at TIMESTAMP DEFAULT NOW()  
);

**campaign\_redemptions**

CREATE TABLE campaign\_redemptions (  
 id UUID PRIMARY KEY,  
 campaign\_id UUID REFERENCES campaigns(id),  
 user\_id UUID REFERENCES users(id),  
 offer\_id VARCHAR(255), -- TLP offer ID  
 points\_redeemed INTEGER,  
 redemption\_date TIMESTAMP DEFAULT NOW(),  
 status VARCHAR(50) DEFAULT 'pending' -- 'pending', 'completed', 'cancelled'  
);

### API Endpoints

#### Authentication & Authorization

POST /api/auth/login - User login (password or OAuth)  
POST /api/auth/logout - User logout  
POST /api/auth/refresh - Refresh authentication token  
GET /api/auth/profile - Get current user profile  
POST /api/auth/register - Register new user (admin only)

#### Campaign Management

POST /api/campaigns - Create new campaign  
GET /api/campaigns - List campaigns  
GET /api/campaigns/:id - Get campaign details  
PUT /api/campaigns/:id - Update campaign  
DELETE /api/campaigns/:id - Delete campaign  
POST /api/campaigns/:id/approve - Approve campaign for execution  
POST /api/campaigns/:id/execute - Execute campaign

#### Schema Management

POST /api/campaigns/:id/schema - Upload database schema  
GET /api/campaigns/:id/schema - Get schema analysis results  
POST /api/campaigns/:id/schema/feedback - Provide schema feedback

#### Rules Engine

POST /api/rules/analyze-schema - Analyze database schema  
POST /api/rules/generate - Generate rules from natural language  
POST /api/rules/validate - Validate generated rules  
POST /api/rules/generate-code - Generate execution code

#### Campaign Execution

GET /api/campaigns/:id/progress - Get campaign progress  
POST /api/campaigns/:id/allocate-points - Allocate points to participants  
GET /api/campaigns/:id/participants - Get participant list

#### Participant Dashboard

GET /api/participants/campaigns - Get user's campaigns  
GET /api/participants/:campaignId/progress - Get personal progress  
GET /api/participants/:campaignId/points - Get point balance  
GET /api/participants/:campaignId/transactions - Get transaction history  
GET /api/participants/:campaignId/offers - Get available redemption offers  
POST /api/participants/:campaignId/redeem - Redeem points for offer

#### TLP Integration

POST /api/tlp/configure - Configure TLP API credentials  
GET /api/tlp/health - Check TLP API connectivity  
POST /api/tlp/generate-graphics - Generate offer graphics

### User Interface Design

#### Admin Dashboard

**Campaign Creation Interface:** -

**Campaign Details Form**: Name, description, start/end dates –

**Goals Configuration**: Individual and regional goal inputs with currency selection –

**Eligibility Criteria Builder**: Natural language input with validation –

**Redemption Prizes**: Prize description, point requirements, auto-generated graphics - **Schema Upload**: Drag-and-drop or file upload for database schema –

**Schema Analysis Feedback**: Visual indicators of AI understanding success rate –

**TLP Configuration**: API key and endpoint URL input fields –

**Backend Connection**: Database connection configuration for rule execution

**Campaign Management Interface:** -

**Campaign List**: Table view with status, progress, and action buttons –

**Campaign Details**: Comprehensive view of all campaign components –

**Approval Workflow**: Review and approve campaigns before execution –

**Execution Monitoring**: Real-time progress tracking and error handling

#### Participant Dashboard

**Campaign Overview:** -

**Active Campaigns**: List of campaigns user is participating in –

**Progress Visualization**: Charts showing goal progress and point accumulation –

**Point Balance**: Current point balance with transaction history –

**Available Offers**: Redemption offers with auto-generated graphics –

**Achievement Badges**: Visual indicators for milestones and achievements

**Interactive Features:** -

**Progress Charts**: Line charts showing progress over time –

**Goal Thermometers**: Visual progress indicators for each goal –

**Transaction History**: Detailed list of point transactions –

**Redemption Interface**: One-click redemption with confirmation dialogs

#### Authentication Interface

**Login Options:** -

**Password Authentication**: Traditional email/password login –

**OAuth Integration**: Google, Microsoft, and other OAuth providers –

**Multi-factor Authentication**: Optional 2FA for enhanced security –

**Role-based Access**: Different interfaces for admins vs participants

### System Workflow

#### 1. Campaign Creation Workflow

**Step 1: Campaign Setup**

1. Admin logs in and navigates to campaign creation

2. Fills in basic campaign details (name, dates, description)

3. Configures goals (individual and regional targets)

4. Defines eligibility criteria in natural language 5. Specifies redemption prizes and point requirements

**Step 2: Schema Upload and Analysis**

1. Uploads database schema (JSON, SQL, or visual schema)

2. AI analyzes schema and provides understanding score

3. Admin reviews analysis and provides feedback if needed

4. System refines understanding based on feedback

5. Final schema understanding is confirmed

**Step 3: TLP Configuration**

1. Admin enters TLP API key and endpoint URL

2. System validates TLP connectivity

3. Configures backend database connection for rule execution

4. Tests connection to source system

**Step 4: Code Generation and Approval**

1. AI generates campaign execution code

2. System creates TLP point types and offers

3. Admin reviews generated code and TLP entities

4. Admin approves campaign for execution

5. System activates campaign and begins monitoring

#### 2. Campaign Execution Workflow

**Step 1:**

**Participant Onboarding**

1. Participants log in to the system

2. System creates TLP member accounts if needed

3. Participants are assigned to campaigns

4. Initial point balances are established

**Step 2: Real-time Monitoring**

1. System continuously monitors source database

2. AI-generated rules execute against live data

3. Points are allocated based on goal achievements

4. Progress is updated in real-time

**Step 3: Redemption Processing**

1. Participants browse available redemption offers

2. System displays auto-generated graphics for each offer

3. Participants redeem points for desired prizes

4. TLP processes redemption transactions

5. Confirmation is sent to participants

#### 3. Analytics and Reporting

**Admin Analytics:** -

Campaign performance metrics

- Participant engagement statistics

- Goal achievement rates

- ROI analysis and reporting

**Participant Analytics:** -

Personal progress tracking

- Historical performance data

- Achievement milestones

- Redemption history

### Security Considerations

1. **API Key Management**: Secure storage and rotation of TLP API keys
2. **Data Encryption**: Encrypt sensitive campaign data at rest
3. **Access Control**: Role-based access to campaign management
4. **Audit Logging**: Track all campaign modifications and executions
5. **Rate Limiting**: Prevent abuse of AI and TLP API calls
6. **OAuth Security**: Secure OAuth integration with proper token handling
7. **Database Security**: Encrypted connections to source systems
8. **Session Management**: Secure session handling with proper timeouts

### Performance Considerations

1. **Caching**: Cache TLP API responses for frequently accessed data
2. **Batch Processing**: Process large campaign executions in batches
3. **Database Indexing**: Optimize queries for campaign tracking
4. **AI Response Caching**: Cache AI-generated code for similar rules

### Monitoring and Analytics

1. **Campaign Performance Metrics**
   * Goal achievement rates
   * Point allocation efficiency
   * Participant engagement
2. **System Health Monitoring**
   * TLP API response times
   * AI model performance
   * Database query performance
3. **Business Intelligence**
   * Campaign ROI analysis
   * Participant behavior patterns
   * Rule effectiveness metrics

### Technical Implementation Details

#### Frontend Architecture (React + TypeScript)

**Component Structure:**

src/  
├── components/  
│ ├── auth/  
│ │ ├── LoginForm.tsx  
│ │ ├── OAuthButton.tsx  
│ │ └── UserProfile.tsx  
│ ├── admin/  
│ │ ├── CampaignForm.tsx  
│ │ ├── SchemaUpload.tsx  
│ │ ├── TLPConfig.tsx  
│ │ └── CampaignList.tsx  
│ ├── participant/  
│ │ ├── Dashboard.tsx  
│ │ ├── ProgressChart.tsx  
│ │ ├── PointBalance.tsx  
│ │ └── RedemptionOffers.tsx  
│ └── shared/  
│ ├── Header.tsx  
│ ├── Sidebar.tsx  
│ └── LoadingSpinner.tsx  
├── services/  
│ ├── api.ts  
│ ├── auth.ts  
│ ├── campaigns.ts  
│ └── tlp.ts  
├── hooks/  
│ ├── useAuth.ts  
│ ├── useCampaigns.ts  
│ └── useTLP.ts  
└── utils/  
 ├── validation.ts  
 ├── formatting.ts  
 └── charts.ts

**Key Technologies:** - **React 18** with TypeScript –

**React Router** for navigation –

**React Query** for data fetching –

**Chart.js** for progress visualizations –

**Material-UI** or **Ant Design** for UI components –

**React Hook Form** for form management

#### Backend Architecture (Node.js + TypeScript)

**Service Structure:**

src/  
├── controllers/  
│ ├── authController.ts  
│ ├── campaignController.ts  
│ ├── participantController.ts  
│ └── tlpController.ts  
├── services/  
│ ├── aiService.ts  
│ ├── campaignService.ts  
│ ├── tlpService.ts  
│ └── graphicsService.ts  
├── middleware/  
│ ├── auth.ts  
│ ├── validation.ts  
│ └── rateLimit.ts  
├── models/  
│ ├── Campaign.ts  
│ ├── User.ts  
│ └── Participant.ts  
└── utils/  
 ├── database.ts  
 ├── encryption.ts  
 └── logger.ts

**Key Technologies:**

- **Express.js** with TypeScript

- **Prisma** or **TypeORM** for database ORM

- **JWT** for authentication - **Passport.js** for OAuth

- **Bull** for job queues

- **Socket.io** for real-time updates

#### AI Integration Architecture

**Claude 3.5 Sonnet Integration:**

interface AIService {  
 analyzeSchema(schema: string): Promise<SchemaAnalysis>;  
 generateRules(requirements: string, schema: SchemaAnalysis): Promise<GeneratedRules>;  
 generateCode(rules: GeneratedRules): Promise<GeneratedCode>;  
 generateGraphics(description: string): Promise<GraphicURL>;  
}  
  
class ClaudeService implements AIService {  
 private client: Anthropic;  
   
 async analyzeSchema(schema: string): Promise<SchemaAnalysis> {  
 const prompt = this.buildSchemaAnalysisPrompt(schema);  
 const response = await this.client.messages.create({  
 model: 'claude-3-5-sonnet-20241022',  
 max\_tokens: 4096,  
 temperature: 0.1,  
 messages: [{ role: 'user', content: prompt }]  
 });  
   
 return this.parseSchemaAnalysis(response.content[0].text);  
 }  
   
 // Additional methods for rule generation and code generation  
}

#### Database Schema Implementation

**Prisma Schema:**

model User {  
 id String @id @default(cuid())  
 email String @unique  
 passwordHash String?  
 firstName String?  
 lastName String?  
 role UserRole @default(PARTICIPANT)  
 oauthProvider String?  
 oauthId String?  
 createdAt DateTime @default(now())  
 updatedAt DateTime @updatedAt  
   
 campaigns UserCampaign[]  
 redemptions CampaignRedemption[]  
}  
  
model Campaign {  
 id String @id @default(cuid())  
 name String  
 description String?  
 startDate DateTime  
 endDate DateTime  
 status CampaignStatus @default(DRAFT)  
 tlpApiKey String?  
 tlpEndpointUrl String?  
 backendConnectionConfig Json?  
 createdById String  
 createdAt DateTime @default(now())  
 updatedAt DateTime @updatedAt  
   
 createdBy User @relation(fields: [createdById], references: [id])  
 rules CampaignRule[]  
 executions CampaignExecution[]  
 schemas CampaignSchema[]  
 userCampaigns UserCampaign[]  
 redemptions CampaignRedemption[]  
}  
  
model CampaignRule {  
 id String @id @default(cuid())  
 campaignId String  
 ruleType RuleType  
 ruleDefinition Json  
 generatedCode String?  
 schemaUnderstandingScore Decimal? @db.Decimal(3, 2)  
 schemaFeedback String?  
 createdAt DateTime @default(now())  
   
 campaign Campaign @relation(fields: [campaignId], references: [id])  
}  
  
enum UserRole {  
 ADMIN  
 PARTICIPANT  
}  
  
enum CampaignStatus {  
 DRAFT  
 APPROVED  
 ACTIVE  
 COMPLETED  
 CANCELLED  
}  
  
enum RuleType {  
 GOAL  
 ELIGIBILITY  
 PRIZE  
}

#### Real-time Features

**WebSocket Implementation:**

// Real-time progress updates  
io.on('connection', (socket) => {  
 socket.on('join-campaign', (campaignId) => {  
 socket.join(`campaign-${campaignId}`);  
 });  
   
 socket.on('disconnect', () => {  
 // Handle disconnection  
 });  
});  
  
// Emit progress updates  
const emitProgressUpdate = (campaignId: string, progress: ProgressData) => {  
 io.to(`campaign-${campaignId}`).emit('progress-update', progress);  
};

#### Graphics Generation Service

**Auto-Generated Graphics:**

class GraphicsService {  
 async generateOfferGraphic(description: string): Promise<string> {  
 // Use AI to generate relevant graphics based on offer description  
 const prompt = `Create a professional graphic for a loyalty campaign offer: ${description}`;  
   
 // Integration with DALL-E or similar image generation service  
 const imageUrl = await this.generateImage(prompt);  
   
 return imageUrl;  
 }  
}

### Implementation Phases – 6 week project

#### Phase 1: Core Infrastructure (1 wees)

* Set up React frontend with TypeScript
* Implement PostgreSQL database with Prisma
* Create basic authentication system (password + OAuth)
* Set up Express.js backend with TypeScript
* Integrate Claude 3.5 Sonnet API

#### Phase 2: Admin Dashboard (1 week)

* Implement campaign creation interface
* Build schema upload and analysis functionality
* Create TLP configuration interface
* Develop campaign management and approval workflow
* Add real-time progress monitoring

#### Phase 3: Rules Engine (1 week)

* Implement schema analysis with AI feedback
* Develop natural language processing for rule definition
* Create code generation for campaign rules
* Build TLP API integration layer
* Add graphics generation for redemption offers

#### Phase 4: Participant Dashboard (1 week)

* Implement participant authentication and onboarding
* Create progress visualization with charts
* Build point balance and transaction history
* Develop redemption interface with auto-generated graphics
* Add real-time updates via WebSocket

#### Phase 5: AI Recommendations & Analytics (1 week)

* Implement AI-powered campaign recommendations
* Create learning system from historical data
* Build comprehensive analytics and reporting
* Add performance optimization features

#### Phase 6: Testing and Deployment (1 week)

* Comprehensive testing of all components
* Performance optimization and security audit
* User acceptance testing
* Production deployment and monitoring setup

### Risk Mitigation

1. **AI Model Reliability**
   * Implement fallback mechanisms for AI failures
   * Human review process for generated code
   * Version control for AI-generated rules
2. **TLP API Dependencies**
   * Implement retry mechanisms for API failures
   * Cache critical data locally
   * Monitor API rate limits
3. **Data Integrity**
   * Implement transaction rollback capabilities
   * Regular data validation checks
   * Backup and recovery procedures

### Conclusion

The Incentiva Loyalty Campaign Management System represents a significant advancement in loyalty program management by combining AI-powered intelligence with robust TLP API integration. The system’s ability to translate natural language requirements into executable loyalty campaigns will democratize sophisticated loyalty program creation while maintaining the reliability and scalability of enterprise-grade systems.

## Appendix A: TLP API Reference

### Point Management APIs

**Create Point Type** - **Endpoint**: POST /points - **Documentation**: <https://exata-customer.pravici.io/documentation/api/points#Create> - **Purpose**: Create new point types for campaign-specific currencies

**Update Point Type** - **Endpoint**: PUT /points/:id - **Documentation**: <https://exata-customer.pravici.io/documentation/api/points#Update> - **Purpose**: Modify existing point type configurations

### Offer Management APIs

**Create Offer** - **Endpoint**: POST /offers - **Documentation**: <https://exata-customer.pravici.io/documentation/api/offers#Create> - **Purpose**: Create accrual and redemption offers for campaigns

**Update Offer** - **Endpoint**: PUT /offers/:id - **Documentation**: <https://exata-customer.pravici.io/documentation/api/offers#Update> - **Purpose**: Modify offer configurations

**Search Offers** - **Endpoint**: GET /offers - **Documentation**: <https://exata-customer.pravici.io/documentation/api/offers#Search> - **Purpose**: Retrieve and filter offers

### Transaction APIs

**Accrue Points** - **Endpoint**: POST /transactions/accrue - **Documentation**: <https://exata-customer.pravici.io/documentation/api/transactions#Accrue> - **Purpose**: Award points to participants based on campaign rules

**Redeem Points** - **Endpoint**: POST /transactions/redeem - **Documentation**: <https://exata-customer.pravici.io/documentation/api/transactions#Redeem> - **Purpose**: Process point redemptions for prizes

**Issue Points** - **Endpoint**: POST /transactions/issue - **Documentation**: <https://exata-customer.pravici.io/documentation/api/transactions#Issue> - **Purpose**: Mint new points for campaign distribution

### Member Management APIs

**Create Member** - **Endpoint**: POST /members - **Documentation**: <https://exata-customer.pravici.io/documentation/api/members#Create> - **Purpose**: Register campaign participants

**Update Member** - **Endpoint**: PUT /members/:id - **Documentation**: <https://exata-customer.pravici.io/documentation/api/members#Update> - **Purpose**: Update participant information

**Search Members** - **Endpoint**: GET /members - **Documentation**: <https://exata-customer.pravici.io/documentation/api/members#Search> - **Purpose**: Find and filter campaign participants

### Tier Management APIs

**Create Tier** - **Endpoint**: POST /tiers - **Documentation**: <https://exata-customer.pravici.io/documentation/api/tiers#Create> - **Purpose**: Create participant tiers for advanced campaigns

**Update Tier** - **Endpoint**: PUT /tiers/:id - **Documentation**: <https://exata-customer.pravici.io/documentation/api/tiers#Update> - **Purpose**: Modify tier configurations

### Partner and Location APIs

**Create Partner** - **Endpoint**: POST /partners - **Documentation**: <https://exata-customer.pravici.io/documentation/api/partners#Create> - **Purpose**: Register business partners for multi-entity campaigns

**Create Location** - **Endpoint**: POST /locations - **Documentation**: <https://exata-customer.pravici.io/documentation/api/locations#Create> - **Purpose**: Register physical locations for location-based campaigns

## Appendix B: AI Integration Specifications

### Claude 3.5 Sonnet Integration

**API Configuration:**

interface ClaudeConfig {  
 apiKey: string;  
 model: 'claude-3-5-sonnet-20241022';  
 maxTokens: 4096;  
 temperature: 0.1; // Low temperature for consistent code generation  
}

**Schema Analysis Prompt:**

Analyze the following database schema and identify:  
1. Tables relevant to sales and customer data  
2. Key relationships between tables  
3. Fields needed for loyalty campaign calculations  
4. Data types and constraints  
  
Schema: [SCHEMA\_JSON]

**Rule Generation Prompt:**

Based on the campaign requirements and database schema, generate:  
1. SQL queries for goal calculation  
2. TypeScript code for TLP API integration  
3. Validation logic for eligibility criteria  
4. Documentation for the generated code  
  
Campaign Requirements: [NATURAL\_LANGUAGE\_RULES]  
Database Schema: [SCHEMA\_ANALYSIS]

### Error Handling and Fallbacks

1. **AI Service Unavailable**
   * Fallback to predefined rule templates
   * Manual rule configuration interface
   * Queue processing for retry
2. **Code Generation Failures**
   * Human review interface
   * Manual code editing capabilities
   * Version control for rule modifications
3. **TLP API Failures**
   * Retry with exponential backoff
   * Local caching of critical data
   * Graceful degradation of features