Netflix Subscription Management System

Project Overview

The Netflix Subscription Management System is a backend service built with Go that manages subscription plans, payment methods, and subscription transactions for a Netflix-like streaming service. The system integrates with Razorpay for payment processing, supports both one-time and recurring subscription plans, and provides a comprehensive API for managing the entire subscription lifecycle.

Architecture

The application follows a clean architecture pattern with clear separation of concerns:

- **Controllers**: Handle HTTP requests and responses
- **Services**: Implement business logic
- **Repositories**: Manage data access and persistence
- **Models**: Define data structures

Tech Stack

- **Backend**: Go with Echo framework
- **Database**: MySQL
- **Payment Processing**: Razorpay API
- **Containerization**: Docker and Docker Compose
- **API Documentation**: Postman Collection

Features

Card Management

- Create payment cards with validation
- Retrieve card details (masked for security)
- Update card information
- Delete cards
- Set default payment card
- List all cards for a user

Subscription Plans

- Multiple subscription tiers (Budget, Standard, Premium)

- Different pricing models (monthly, yearly)
- Various feature attributes (number of screens, video quality)

Subscription Management

- Create new subscriptions
- View active subscription
- View subscription history
- Stop active subscription
- Support for auto-renewal

Payment Processing

- One-time payments
- Recurring subscription payments
- Payment verification
- Webhook handling for payment events

Database Schema

Cards Table

```
- id (VARCHAR(36), Primary Key)
- user_id (VARCHAR(36))
- card_number (VARCHAR(16))
- card_holder_name (VARCHAR(100))
- expiry_month (INT)
- expiry_year (INT)
- card_type (VARCHAR(20))
- last_four_digits (VARCHAR(4))
- is_default (BOOLEAN)
- created_at, updated_at (TIMESTAMP)
```

Subscription Products Table

```
- id (VARCHAR(36), Primary Key)
- name (VARCHAR(100))
- created_at, updated_at (TIMESTAMP)
```

Subscription Product Attributes Table

- id (VARCHAR(36), Primary Key)
- product_id (VARCHAR(36), Foreign Key)
- name (VARCHAR(100))
- value (VARCHAR(100))
- created_at, updated_at (TIMESTAMP)

Subscription Plans Table

- id (VARCHAR(36), Primary Key)
- product_id (VARCHAR(36), Foreign Key)
- name (VARCHAR(100))
- price_monthly (DECIMAL(10,2))
- price_yearly (DECIMAL(10,2))
- created_at, updated_at (TIMESTAMP)

Subscription Transactions Table

- id (VARCHAR(36), Primary Key)
- user_id (VARCHAR(36))
- product_id (VARCHAR(36), Foreign Key)
- plan_id (VARCHAR(36), Foreign Key)
- card_id (VARCHAR(36), Foreign Key)
- is_renewal (BOOLEAN)
- is_active (BOOLEAN)
- payment_type (VARCHAR(20))
- amount (DECIMAL(10,2))
- start_date, end_date, next_renewal_date (TIMESTAMP)
- razorpay_order_id, razorpay_payment_id, razorpay_subscription_id (VARCHAR(100))
- auto_renewal (BOOLEAN)
- created_at, updated_at (TIMESTAMP)

API Endpoints

Card Management

Method	Endpoint	Description
POST	/api/cards	Create a new card
GET	/api/cards?userId={userId}	Get all cards for a user
GET	/api/cards/{id}?userId={userId}	Get a specific card
PUT	/api/cards/{id}	Update a card
DELETE	/api/cards/{id}?userId={userId}	Delete a card
PUT	/api/cards/{id}/default?userId={userId}	Set a card as default
DELETE	/api/cards/all?userId={userId}	Delete all cards for a user

Subscription Management

Method	Endpoint	Description
GET	/api/subscriptions/plans	Get all available plans
GET	/api/subscriptions/active?userId={userId}	Get active subscription for a user
GET	/api/subscriptions/history?userId={userId}	Get subscription history
POST	/api/subscriptions	Create a new subscription
PUT	/api/subscriptions/{id}/renew?userId={userId}	Renew a subscription
PUT	/api/subscriptions/{id}/stop?userId={userId}	Stop a subscription

Payment Integration

Method	Endpoint	Description
GET	/api/subscriptions/test-razorpay	Test Razorpay connection
POST	/api/subscriptions/verify-payment	Verify a payment
POST	/webhooks/razorpay	Handle Razorpay webhook events

System

Method	Endpoint	Description
GET	/health	Health check endpoint

Deployment

The application is containerized using Docker and can be deployed using Docker Compose.

Docker Compose Configuration

```yaml
version: '3.8'

services:
  mysql:
    image: mysql:8.0
    container_name: subscription-db
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: subscription_db
      MYSQL_USER: appuser
      MYSQL_PASSWORD: apppassword
    ports:
      - "3306:3306"
    volumes:
      - mysql-data:/var/lib/mysql
      - ./migrations:/docker-entrypoint-initdb.d

  app:
    build: .
    container_name: subscription-api
    restart: always
    depends_on:
      - mysql
    environment:
      DB_HOST: mysql
      DB_PORT: 3306
      DB_USER: appuser
      DB_PASSWORD: apppassword
      DB_NAME: subscription_db
      RAZORPAY_KEY_ID: ${RAZORPAY_KEY_ID}
      RAZORPAY_KEY_SECRET: ${RAZORPAY_KEY_SECRET}
      RAZORPAY_WEBHOOK_SECRET: ${RAZORPAY_WEBHOOK_SECRET}
    ports:
      - "8080:8080"

volumes:
  mysql-data:
```

Testing

All API endpoints have been thoroughly tested using Postman. The following test scenarios were executed:

1. Card Management
    - Creating cards with valid and invalid information
    - Retrieving, updating, and deleting cards
    - Setting default cards
2. Subscription Management
    - Retrieving available plans
    - Creating subscriptions with different plan types
    - Stopping active subscriptions
    - Viewing subscription history
3. Payment Processing
    - Testing Razorpay connection
    - Simulating one-time payments using a custom HTML page and Razorpay's domestic test card
    - Testing payment verification
    - Simulating webhook events for various payment scenarios

Razorpay Integration Details

The system integrates with Razorpay for payment processing, supporting both one-time and recurring payment models:

- **One-time Payments**: Creates an order in Razorpay and updates the transaction when payment is completed
- **Recurring Subscriptions**: Creates a subscription in Razorpay that automatically charges the customer at regular intervals

Webhook Handling

The system implements webhook handlers for the following Razorpay events:

- `payment.authorized`: Updates the transaction when a payment is successful
- `subscription.charged`: Creates a renewal transaction when a subscription payment is processed
- `subscription.cancelled`: Marks the subscription as inactive
- `payment.failed`: Logs payment failures

Payment Simulation

For testing purposes, a custom HTML page was created to simulate the payment flow using Razorpay's checkout integration. The payment was successfully processed using Razorpay's domestic test card.

User Flows

Creating a Subscription

1. User adds a payment card
2. User selects a subscription plan (Budget, Standard, Premium)
3. User chooses payment frequency (monthly or yearly)
4. User decides on auto-renewal option
5. System creates a subscription record
6. For one-time payments:
   - System creates a Razorpay order
   - User completes payment through Razorpay checkout
   - Razorpay sends a webhook notification
   - System updates the subscription status
7. For recurring subscriptions:
   - System creates a Razorpay subscription
   - User authorizes the recurring payment
   - Razorpay handles future billing cycles

Stopping a Subscription

1. User requests to stop an active subscription
2. System marks the subscription as inactive
3. If auto-renewal is enabled, system cancels the subscription in Razorpay
4. User retains access until the end of the current billing period

Security Considerations

- Card numbers are masked in API responses
- Only the last four digits of card numbers are stored in plain text
- All API endpoints require user authentication
- Webhook endpoints verify Razorpay signatures to prevent tampering

Conclusion

The Netflix Subscription Management System provides a comprehensive backend solution for managing subscription-based services. The system's modular architecture allows for easy maintenance and extension, while the Razorpay integration provides a secure and reliable payment processing solution.