Edviron Payments System

Project Documentation

Author: [Your Name]

Date: September 2025

Version: 1.0

Table of Contents

---

1. Project Overview

1.1 Introduction

The Edviron Payments System is a microservices-based payment platform built for secure fee collection, payment tracking, and comprehensive admin/student dashboards. It demonstrates modern distributed system architecture with event-driven communication.

1.2 Key Objectives

- Streamline fee collection for educational institutions
- Provide real-time payment tracking and analytics
- Ensure secure and compliant payment processing
- Enable comprehensive transaction management

1.3 Target Users

- **Students**: Payment requests, transaction history
- **Administrators**: Payment management, analytics
- **System Admins**: Configuration, monitoring

---

2. System Architecture

2.1 Architecture Overview

Microservices architecture with 4 core services:

- **Auth Service** (Port 3000): Authentication & authorization
- **Payment Service** (Port 3001): Payment processing
- **Transaction Service** (Port 3004): Transaction management
- **Webhook Service** (Port 3003): Webhook handling

2.2 Infrastructure Components

- **Kong API Gateway**: Centralized routing and security
- **Apache Kafka**: Event-driven communication
- **MongoDB**: Document database storage

2.3 Event Flow

1. Student creates payment → Payment Service

2. `payment.created` event → Kafka → Transaction Service
3. Gateway processes payment → Webhook Service
4. `payment.updated` event → Real-time dashboard updates

---

3. Technology Stack

3.1 Backend

- **Runtime**: Node.js 18+, Express.js, ES Modules
- **Database**: MongoDB 6.0+ with Mongoose ODM
- **Message Queue**: Apache Kafka 3.0+ (KRaft mode)
- **Authentication**: JWT (Access + Refresh tokens)
- **API Gateway**: Kong Gateway with Docker

3.2 Frontend

- **Framework**: React 18, TypeScript
- **Build Tool**: Vite
- **Styling**: TailwindCSS
- **State Management**: React Query, Context API
- **Charts**: Recharts

3.3 DevOps

- **Containerization**: Docker, Docker Compose
- **Utilities**: p-retry, kafkajs, dotenv, winston

---

4. Features

4.1 Student Portal

- **Authentication**: JWT-based login/registration
- **Payment Requests**: Create payment links
- **Gateway Integration**: Seamless payment redirection
- **Transaction History**: Real-time status tracking

4.2 Admin Dashboard

- **Analytics**: Transaction metrics by status, gateway, date ranges
- **Search**: Advanced filtering with multiple criteria
- **Status Management**: Real-time order verification

4.3 System Features

- **Microservices**: Isolated, scalable services
- **Event-Driven**: Kafka-based async messaging
- **Data Consistency**: Idempotent transaction processing
- **Webhooks**: Secure signature verification

---

5. API Documentation

5.1 Authentication

```http
POST /api/auth/register - User registration
POST /api/auth/login - User login
POST /api/auth/refresh - Refresh JWT token
POST /api/auth/logout - User logout
```

5.2 Payments

```http
POST /api/payments/create - Create payment request
GET /api/payments/:orderId/status - Get payment status
```

5.3 Transactions

```http
GET /api/transactions - List transactions (supports filtering)
GET /api/transactions/:orderId - Get specific transaction
```

5.4 Webhooks

```http
```

```
POST /api/webhook/payment - Payment provider callback
```

5.5 Sample Request/Response

Create Payment:

```json
// Request
{
  "school_id": "65b0e6293e9f76a9694d84b4",
  "amount": "100",
  "callback_url": "http://localhost:5173/payments/callback/success"
}

// Response
{
  "success": true,
  "data": {
    "collect_request_id": "6808bc4888e4e3c149e757f1",
    "collect_request_url": "https://payments.edviron.com/...",
    "sign": "jwt_signature"
  }
}
```

---

6. Database Design

6.1 Collections Overview

- **Users**: Authentication and profile data
- **Orders**: Payment request information
- **Transactions**: Payment processing records
- **Webhook_Logs**: External webhook data

6.2 Key Schemas

Users Collection:

```javascript
```

```
{
  _id: ObjectId,
  email: String (unique),
  password: String (hashed),
  name: String,
  role: String,
  school_id: ObjectId,
  created_at: Date,
  updated_at: Date
}
```

Orders Collection:

```javascript
{
  _id: ObjectId,
  order_id: String (unique),
  school_id: ObjectId,
  user_id: ObjectId,
  amount: Number,
  status: String,
  payment_link: Object,
  created_at: Date,
  updated_at: Date
}
```

---

7. Project Structure

```
Edviron_Payment_Integration/
├── Backend/
│   ├── services/
│   │   ├── auth-service/          # Authentication
│   │   ├── payment-service/       # Payment processing
│   │   ├── transaction-service/   # Transaction management
│   │   └── webhook-service/       # Webhook handling
│   └── infra/
│       ├── kong/                  # Kong configuration
│       └── docker-compose.yaml    # Infrastructure setup
├── Frontend/
│   ├── src/
```

```
│   │   ├── components/          # Reusable components
│   │   ├── pages/               # Page components
│   │   ├── services/            # API services
│   │   └── contexts/            # React contexts
│   └── package.json
└── README.md
```

---

8. Installation & Setup

8.1 Prerequisites

- Node.js 18+
- MongoDB 6.0+
- Docker & Docker Compose
- Apache Kafka

8.2 Quick Start

bash

```
# 1. Clone repository
git clone https://github.com/sanjay-yadav-05/Edvrion_Payment_Integration_and-_Transaction_Analysis_Assessment.git

# 2. Install dependencies
./scripts/install-all.sh

# 3. Setup environment files
cp services/auth-service/.env.example services/auth-service/.env
# ... repeat for other services

# 4. Start infrastructure
docker run -d --name mongodb -p 27017:27017 mongo:6.0
cd infra && docker compose up -d

# 5. Start Kafka
bin/kafka-server-start.sh config/server.properties

# 6. Start services
cd Backend/services/auth-service && npm run dev &
cd Backend/services/payment-service && npm run dev &
# ... start other services
```

8.3 Access Points

- **API Gateway**: http://localhost:8000/api/
- **Auth Service**: http://localhost:3000
- **Payment Service**: http://localhost:3001
- **Transaction Service**: http://localhost:3004
- **Webhook Service**: http://localhost:3003

---

9. Configuration

9.1 Environment Variables

Each service requires specific environment configuration:

```
env
```

```
# Server Configuration
PORT=3001
NODE_ENV=development

# Database
MONGO_URI=mongodb://127.0.0.1:27017/Edviron_DB

# JWT Configuration
JWT_ACCESS_SECRET=your_access_secret_here
JWT_REFRESH_SECRET=your_refresh_secret_here

# Kafka Configuration
KAFKA_BROKERS=localhost:9092
KAFKA_CLIENT_ID=payment-service

# Payment Gateway
PG_API_URL=https://api.edviron.com
PG_API_KEY=your_api_key_here
PG_WEBHOOK_SECRET=your_webhook_secret
```

---

10. Testing

10.1 Testing Strategy

bash

```
# Run all tests
npm run test

# Service-specific tests
cd services/auth-service && npm run test

# Integration tests
npm run test:integration
```

10.2 Manual Testing Checklist

- ☐ User authentication flow
- ☐ Payment creation and processing
- ☐ Webhook handling

- ☐ Transaction filtering and search
- ☐ Real-time status updates

---

11. Development Guidelines

11.1 Code Standards

- ESLint configuration for code quality
- Prettier for consistent formatting
- Conventional commits for version control
- Comprehensive error handling

11.2 Adding Features

1. Create feature branch
2. Implement in relevant microservice
3. Add tests and documentation
4. Submit pull request

---

12. Security

12.1 Authentication & Authorization

- JWT-based authentication with refresh tokens
- Bcrypt password hashing with salt rounds
- Role-based access control (RBAC)
- Protected routes and API endpoints

12.2 Data Security

- Webhook signature verification
- Input validation and sanitization
- Secure environment variable management
- HTTPS enforcement in production

---

13. Performance

13.1 Database Optimization

- Optimized indexes for frequent queries
- Connection pooling for efficiency
- Query performance monitoring
- Data aggregation for analytics

13.2 Application Performance

- Caching strategies for frequently accessed data
- Asynchronous processing with Kafka
- Connection pooling and resource management
- Load balancing through Kong Gateway

---

14. Monitoring

14.1 Health Checks

- `/health` endpoints on each service
- Service availability monitoring
- Database connection health
- Kafka broker connectivity

14.2 Logging

- Winston-based structured logging
- Request/response logging with Morgan
- Error tracking and alerting
- Performance metrics collection

---

15. Troubleshooting

15.1 Common Issues

Database Connection Failed:

- Verify MongoDB URI and credentials
- Check network connectivity

- Ensure MongoDB service is running

JWT Authentication Errors:

- Validate JWT secret configuration
- Check token expiration settings
- Verify token format in requests

Payment Gateway Issues:

- Verify API credentials and endpoints
- Check webhook signature verification
- Validate payload structure

Kafka Connection Problems:

- Ensure Kafka broker is running
- Verify broker configuration
- Check topic creation and permissions

---

16. Deployment

16.1 Production Checklist

- ☐ Environment variables configured
- ☐ Database indices optimized
- ☐ SSL certificates installed
- ☐ Rate limiting configured
- ☐ Monitoring and alerting setup
- ☐ Backup strategies implemented
- ☐ Load balancing configured

16.2 Docker Deployment

```bash
```

```
# Build and deploy with Docker Compose
docker-compose up -d --build

# Scale specific services
docker-compose up -d --scale payment-service=3
```

---

17. Maintenance

17.1 Regular Tasks

- Monitor application performance metrics
- Review and update dependencies
- Database backup and maintenance
- Security patch updates
- Log rotation and cleanup

17.2 Monitoring Setup

- Application performance monitoring (APM)
- Database performance tracking
- Error tracking and alerting
- User activity analytics
- API usage monitoring

---

18. Future Enhancements

18.1 Planned Features

- Multi-tenant architecture support
- Advanced analytics dashboard
- Mobile application development
- Additional payment gateway integrations
- Automated reconciliation system

18.2 Technical Improvements

- GraphQL API implementation

- Service mesh architecture
- Advanced caching strategies
- Machine learning for fraud detection
- Blockchain integration for transparency

---

19. Contributing

19.1 Development Process

1. Fork the repository
2. Create feature branch (`git checkout -b feature/AmazingFeature`)
3. Commit changes (`git commit -m 'Add AmazingFeature'`)
4. Push to branch (`git push origin feature/AmazingFeature`)
5. Open Pull Request

19.2 Code Review Guidelines

- Comprehensive code reviews required
- Automated testing must pass
- Documentation updates required
- Security considerations addressed

---

20. License & Support

20.1 License

20.2 Acknowledgments

- Built as part of the Edviron technical assessment
- Demonstrates modern microservices architecture patterns
- Implements event-driven design principles

20.3 Support

For questions or support:

- Open an issue on GitHub
- Contact the development team
- Check documentation at https://docs.edviron.com

Project Statistics:

- **Total Services**: 4 microservices
- **Database Collections**: 6 main collections
- **API Endpoints**: 15+ REST endpoints
- **Technology Stack**: 20+ technologies
- **Documentation Pages**: 25+ comprehensive sections

Development Team Contact:

- **Email**: support@edviron.com
- **GitHub**: https://github.com/sanjay-yadav-05/Edviron_Payment_Integration
- **Documentation**: https://docs.edviron.com

This documentation serves as a comprehensive guide for developers, administrators, and stakeholders involved in the Edviron Payments System project. For the most up-to-date information, please refer to the project repository and official documentation.