**PayPlus Payment Gateway - Complete Documentation**

**📋 Project Overview**

**PayPlus** is a secure payment processing system built with microservices architecture, simulating traditional on-premise deployment using AWS EC2 instances.

**🏗️ Architecture**

**Infrastructure Setup**

text

EC2 Instance 1: app-server (Backend Microservices)

├── Gateway Service: 8080

├── Auth Service: 8081

├── Payment Service: 8082

└── Token Service: 8083

EC2 Instance 2: db-server (MySQL Database)

├── MySQL: 3306

└── Database: payplus

EC2 Instance 3: web-server (Frontend + Nginx)

├── React Frontend

└── Nginx: 80

**Technology Stack**

* **Backend**: Spring Boot 3.1.0, Java 17
* **Frontend**: React 18, Vite
* **Database**: MySQL 8.0
* **Web Server**: Nginx
* **Infrastructure**: AWS EC2 Ubuntu 22.04

**📁 Complete File Structure**

**Backend Services**

text

backend/

├── gateway-service/

│ ├── pom.xml

│ ├── src/main/java/com/payplus/GatewayApplication.java

│ └── src/main/resources/application.yml

├── auth-service/

│ ├── pom.xml

│ ├── src/main/java/com/payplus/AuthApplication.java

│ ├── src/main/java/com/payplus/controller/AuthController.java

│ └── src/main/resources/application.yml

├── payment-service/

│ ├── pom.xml

│ ├── src/main/java/com/payplus/PaymentApplication.java

│ ├── src/main/java/com/payplus/controller/PaymentController.java

│ └── src/main/resources/application.yml

└── token-service/

├── pom.xml

├── src/main/java/com/payplus/TokenApplication.java

├── src/main/java/com/payplus/controller/TokenController.java

└── src/main/resources/application.yml

**Frontend Application**

text

frontend/

├── package.json

├── vite.config.js

├── index.html

└── src/

├── index.jsx

├── App.jsx

├── App.css

├── index.css

├── components/

│ ├── Header.jsx

│ └── Footer.jsx

└── pages/

├── Dashboard.jsx

├── Payment.jsx

├── Tokenization.jsx

└── Authorization.jsx

**🔧 Installation & Deployment**

**Phase 1: Infrastructure Setup**

1. **Create 3 EC2 instances** (t3.medium for app/db, t3.small for web)
2. **Configure Security Groups**:
   * app-server: Allow 8080-8083 from web-server
   * db-server: Allow 3306 from app-server
   * web-server: Allow 80 from anywhere

**Phase 2: Database Setup**

bash

*# On db-server*

sudo apt update

sudo apt install mysql-server -y

sudo mysql\_secure\_installation

*# Create database and user*

mysql -u root -p

CREATE DATABASE payplus;

CREATE USER 'payplus\_user'@'%' IDENTIFIED BY 'payplus\_pass123';

GRANT ALL PRIVILEGES ON payplus.\* TO 'payplus\_user'@'%';

FLUSH PRIVILEGES;

EXIT;

*# Configure remote access*

sudo nano /etc/mysql/mysql.conf.d/mysqld.cnf

*# Change: bind-address = 0.0.0.0*

sudo systemctl restart mysql

**Phase 3: Backend Deployment**

bash

*# On app-server*

sudo apt update

sudo apt install openjdk-17-jdk maven -y

*# Build services*

cd backend

for service in gateway-service auth-service payment-service token-service; do

cd $service

mvn clean package -DskipTests

cd ..

done

*# Start services*

./start-production.sh

**Phase 4: Frontend Deployment**

bash

*# On web-server*

sudo apt update

sudo apt install nodejs npm nginx -y

*# Build frontend*

cd frontend

npm install

npm run build

*# Deploy to Nginx*

sudo cp -r dist/\* /var/www/html/

*# Configure Nginx*

sudo nano /etc/nginx/sites-available/default

**⚙️ Configuration Files**

**Gateway Service (application.yml)**

yaml

server:

port: 8080

spring:

cloud:

gateway:

routes:

- id: auth-service

uri: http://localhost:8081

predicates:

- Path=/api/auth/\*\*

filters:

- StripPrefix=1

- id: payment-service

uri: http://localhost:8082

predicates:

- Path=/api/payment/\*\*

filters:

- StripPrefix=1

- id: token-service

uri: http://localhost:8083

predicates:

- Path=/api/token/\*\*

filters:

- StripPrefix=1

**Auth Service (application.yml)**

yaml

server:

port: 8081

spring:

datasource:

url: jdbc:mysql://DB\_SERVER\_IP:3306/payplus?useSSL=false

username: payplus\_user

password: payplus\_pass123

jpa:

hibernate:

ddl-auto: update

**Nginx Configuration**

nginx

server {

listen 80;

server\_name \_;

root /var/www/html;

index index.html;

location / {

try\_files $uri $uri/ /index.html;

}

location /api/ {

proxy\_pass http://APP\_SERVER\_IP:8080;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

}

**🔄 API Endpoints**

**Authentication Service**

* POST /api/auth/login - User authentication
* POST /api/auth/validate - Token validation

**Payment Service**

* POST /api/payment/process - Process payment
* GET /api/payment/history - Payment history

**Tokenization Service**

* POST /api/token/create - Tokenize card data

**🚀 Startup Scripts**

**Backend Services (start-production.sh)**

bash

#!/bin/bash

echo "Starting PayPlus Microservices..."

pkill -f java

sleep 3

java -jar auth-service/target/auth-service-1.0.0.jar > auth.log 2>&1 &

java -jar payment-service/target/payment-service-1.0.0.jar > payment.log 2>&1 &

java -jar token-service/target/token-service-1.0.0.jar > token.log 2>&1 &

sleep 15

java -jar gateway-service/target/gateway-service-1.0.0.jar > gateway.log 2>&1 &

sleep 10

echo "Services started successfully!"

**🐛 Troubleshooting Guide**

**Common Issues & Solutions**

1. **Database Connection Failed**
   * Check MySQL is running: sudo systemctl status mysql
   * Verify security groups allow port 3306
   * Test connection: mysql -h DB\_IP -u user -p
2. **Build Failures**
   * Ensure Node.js ≥16: node --version
   * Clear cache: rm -rf node\_modules package-lock.json
   * Use .jsx extension for JSX files
3. **Nginx Installation Issues**
   * Clear locks: sudo rm -f /var/lib/dpkg/lock\*
   * Fix packages: sudo dpkg --configure -a
   * Use dev server as backup: npm run dev -- --host 0.0.0.0 --port 80
4. **Service Communication**
   * Test connectivity: nc -zv IP PORT
   * Check logs: tail -f \*.log
   * Verify security group rules

**📊 Monitoring & Logs**

**Key Log Files**

* **Nginx**: /var/log/nginx/access.log, /var/log/nginx/error.log
* **Backend Services**: ~/backend/\*.log
* **System**: /var/log/syslog

**Health Checks**

bash

*# Backend health*

curl http://localhost:8080/actuator/health

*# Database connection*

mysql -h DB\_IP -u user -p -e "SELECT 1"

*# Frontend serving*

curl http://WEB\_SERVER\_IP/

**🔒 Security Considerations**

1. **Network Security**
   * Use security groups to restrict access
   * Consider VPC peering for private communication
   * Implement SSL/TLS in production
2. **Application Security**
   * Implement proper JWT validation
   * Use environment variables for credentials
   * Regular dependency updates
3. **Database Security**
   * Strong passwords for database users
   * Regular backups
   * Network isolation

**🚀 Production Recommendations**

1. **High Availability**
   * Use multiple instances with load balancers
   * Implement database replication
   * Set up auto-scaling groups
2. **Monitoring**
   * Implement CloudWatch monitoring
   * Set up application performance monitoring
   * Configure alerting
3. **CI/CD Pipeline**
   * Automated testing and deployment
   * Blue-green deployment strategy
   * Infrastructure as Code (Terraform)

**📞 Support & Maintenance**

**Regular Maintenance Tasks**

* Update dependencies monthly
* Review security patches
* Monitor performance metrics
* Backup database regularly

**Emergency Procedures**

1. Service failure: Restart using startup scripts
2. Database issues: Restore from latest backup
3. Security breach: Rotate credentials immediately

**Document Version**: 1.0  
**Last Updated**: October 2024  
**Project Status**: Production Ready

**😫 MAJOR PAIN POINTS & SOLUTIONS**

**1. 🐛 Build Process Confusion**

**Pain**: "Why does my code work in VS Code but fail on server?"

**Key Learning**:

text

LOCAL (VS Code) ≠ SERVER (Linux)

├── Node.js versions differ

├── File permissions matter

├── Hidden characters break things

└── Dependencies must match exactly

**Future Fix**:

* Always check node --version on both environments
* Use .jsx extension for JSX files on server
* Test build process BEFORE deployment

**2. 🔗 Network Connectivity Issues**

**Pain**: "Why can't my services talk to each other?"

**Key Learning**:

text

AWS EC2 ≠ Localhost

├── Security Groups = Firewall rules

├── Private IPs for internal communication

├── Public IPs only for external access

└── Ports must be explicitly opened

**Future Fix**:

bash

*# Always test connectivity*

ping TARGET\_IP

nc -zv TARGET\_IP PORT

curl http://TARGET\_IP:PORT/health

**3. 📦 Dependency Hell**

**Pain**: "Why can't I just install Nginx? What are these dpkg errors?"

**Key Learning**:

text

Ubuntu Package System:

apt (user command) → dpkg (core installer) → System files

↓

Background services can LOCK the system:

- unattended-upgrades (auto security updates)

- apt-daily (auto refresh)

- snapd (snap packages)

**Future Fix**:

bash

*# When packages fail:*

sudo killall apt dpkg -9

sudo rm -f /var/lib/dpkg/lock\*

sudo dpkg --configure -a

sudo apt --fix-broken install

**4. 🗂️ File Structure Confusion**

**Pain**: "Which file does what? How are they connected?"

**Key Learning**:

text

React Frontend = Component Tree

├── index.html → Loads → index.jsx → Renders → App.jsx

│ ↓

├── Pages (Dashboard, Payment, etc.)

│ ↓

└── Components (Header, Footer, etc.)

Spring Boot Backend = Service Mesh

├── Gateway → Routes → Services

│ ↓

├── Auth Service (8081)

├── Payment Service (8082)

└── Token Service (8083)

**5. 🔄 API Flow Confusion**

**Pain**: "How does frontend actually reach backend?"

**Key Learning**:

text

User → Frontend (Nginx:80) → API Call → Nginx Proxy → Gateway (8080) → Microservice

↓

React fetch('/api/auth/login') → Nginx sees /api/\* → Forwards to Gateway → Auth Service

**Remember**: Frontend NEVER talks directly to microservices, only through Gateway/Nginx.

**🚨 CRITICAL "AH-HA!" MOMENTS**

**1. JSX File Extension Issue**

bash

*# WRONG: File named .js but contains JSX*

src/index.js (with JSX inside) → BUILD FAILS

*# RIGHT: File named .jsx for JSX content*

src/index.jsx (with JSX inside) → BUILD WORKS

**2. Database Connection Strings**

yaml

*# WRONG: Using localhost from different server*

url: jdbc:mysql://localhost:3306/db

*# RIGHT: Use actual server IP*

url: jdbc:mysql://172.31.32.191:3306/db

**3. Service Startup Order**

bash

*# WRONG: Start gateway first*

Gateway → (fails) → Backend services not ready

*# RIGHT: Start backend first, then gateway*

Auth, Payment, Token → (wait) → Gateway

**💡 FUTURE PROJECT CHEAT SHEET**

**Deployment Checklist:**

1. ✅ Check Node.js versions match
2. ✅ Test database connectivity
3. ✅ Verify security groups allow traffic
4. ✅ Build frontend locally first
5. ✅ Start backend services before gateway
6. ✅ Test API endpoints individually
7. ✅ Check Nginx configuration syntax

**Troubleshooting Flow:**

text

Service Down?

↓

Check if process running: `ps aux | grep java`

↓

Check logs: `tail -f service.log`

↓

Test connectivity: `curl http://localhost:PORT/health`

↓

Check network: `nc -zv IP PORT`

↓

Verify dependencies: `mvn clean package` or `npm install`

**Quick Debug Commands:**

bash

*# Network*

ping TARGET\_IP

nc -zv TARGET\_IP PORT

*# Services*

ps aux | grep java

systemctl status nginx

*# Logs*

tail -f /var/log/nginx/error.log

journalctl -u nginx -f

*# Build*

mvn clean package -DskipTests

npm run build

**🎯 BIGGEST TAKEAWAYS**

**1. Infrastructure First, Code Second**

* Set up ALL servers and networking BEFORE writing code
* Test connectivity between EVERY component
* Document IP addresses and ports

**2. Build Locally, Deploy Remotely**

* Never write code directly on server
* Use Git for everything
* Test build process in clean environment

**3. Microservices = Separate Entities**

* Each service has its own database connection
* Gateway routes traffic, doesn't process data
* Services communicate via APIs, not direct calls

**4. AWS Reality Check**

* Security Groups are your firewall
* Private IPs for internal communication
* Everything is locked down by default

**📚 FOR NEXT PROJECT - REMEMBER:**

**"If it works locally but fails on server, check:**

1. **Versions** (Node, Java, etc.)
2. **Network** (IPs, ports, security groups)
3. **Files** (extensions, permissions, hidden chars)
4. **Dependencies** (package.json, pom.xml)
5. **Order** (start databases before apps)\*\*"

**This documentation will save you HOURS of debugging in future projects!** 🚀