partnr

Build Production-Ready Payment Gateway with Async Processing and Webhooks

Back

Mandatory Task

Domain

Backend Development

Skills

API Development
Communication
Database Management
Deep Learning
Docker Compose Configuration
EDA
Exponential Backoff Implementation
HMAC Signature Generation
Management
Product
Public Relations
SDK Development
Solution Architecture
System Reliability
Webhook Implementation

Difficulty

Advanced

Tools

Bull
Celery
Docker
Express.Js
FastAPI
Go
Java
Node.Js
PostgreSQL
Python
React
Redis
RQ
Spring Boot
Webpack

Industries

Fintech

Submission Received

Your submission has been received and is pending review. You can still update your submission till the effective deadline.

Submitted on 14 Jan 2026, 09:40 pm

Deadline: **21 Jan 2026, 04:59 pm**

Overview | Instructions | Resources | Submit

partnr

Submission Instructions

Submit a fully functional payment gateway application with async processing capabilities:

Required Artifacts:

1. **Working Application**: Complete source code with all services (API, worker service, frontend dashboard, checkout page, embeddable SDK) that can be started with `docker-compose up -d`
2. **Repository URL**: GitHub/GitLab repository with all code
3. **README.md**: Comprehensive documentation including:
   - Setup instructions
   - API endpoint documentation
   - Environment variable configuration
   - Testing instructions
   - Webhook integration guide
   - SDK integration guide
4. **submission.yml** (MANDATORY): Configuration file for automated evaluation containing:
   - Setup commands (install dependencies, configure environment)
   - Start commands (launch application via docker-compose including Redis and worker service)
   - Test commands (run test suite if available)
   - Verify commands (API contract validation, health checks, job queue status)
   - Shutdown commands (gracefully stop services)

Optional Artifacts (Bonus Points):

- Architecture diagram showing async processing flow, job queue architecture, and webhook delivery system
- API documentation (OpenAPI/Swagger spec)
- Video demo showing end-to-end payment flow with webhook delivery and SDK integration
- Screenshots of dashboard webhook configuration and logs

Evaluation Process: Your submission will be evaluated through automated tests that verify:

- All API endpoints work correctly with exact request/response formats
- Database schema matches specifications (refunds, webhook_logs, idempotency_keys tables)
- Frontend pages include required data-test-id attributes
- Docker services start successfully (including Redis and worker service)
- Async payment processing via job queues
- Webhook delivery with HMAC signature verification

- Idempotency key handling
- Refund processing logic
- Embeddable SDK functionality
- Job queue status endpoint

Ensure your application starts successfully with `docker-compose up -d` and all services are accessible on the specified ports (8000 for API, 3000 for dashboard, 3001 for checkout, 6379 for Redis).

Evaluation Overview

Your payment gateway implementation with async processing will be evaluated through multiple methods to assess functionality, code quality, and system design:

Automated Testing: We will run automated tests that verify all API endpoints work correctly with exact request/response formats specified in the requirements. Tests will check database schema compliance (refunds, webhook_logs, idempotency_keys tables), async payment processing via job queues, webhook delivery with HMAC signature verification, idempotency key handling, refund processing logic, embeddable SDK functionality, and job queue status endpoint. Your application must start successfully with `docker-compose up -d` including Redis and worker services, and pass all automated endpoint tests.

Code Review: Your code will be reviewed for architecture quality, async processing patterns, job queue implementation, webhook delivery system, security best practices (HMAC signatures), error handling, and code organization. We will assess how you structure worker services, implement retry logic, handle idempotency, manage refund processing, and build the embeddable SDK. Code should demonstrate clean architecture, proper async patterns, and adherence to security principles.

System Design Assessment: Your submission questionnaire responses will be evaluated to understand your architectural decisions, trade-offs considered, scaling strategies, and engineering judgment. We will assess your understanding of async processing patterns, webhook delivery mechanisms, idempotency implementation, and system reliability considerations.

Human Evaluation: Visual artifacts including architecture diagrams, dashboard screenshots, and video demos will be reviewed for clarity, completeness, and presentation quality. Documentation will be assessed for comprehensiveness and usability.

partnr

About Us

Contact Us

Privacy Policy

Terms and Conditions