EDVIRON — Backend Implementation Summary

A concise, human-friendly summary of the backend work completed, how to test it, and relevant code snippets.

# 1. Overview

* This document summarizes the backend I built for the EDVIRON assessment: authentication, payment gateway integration (create collect request + status), webhook handling, order & order status models, and transaction endpoints. It includes the key files, sample .env, snippets, and testing steps.

# 2. Completed Items

* Project scaffold: routes, controllers, models, middleware, services.
* Authentication: register & login with JWT and auth middleware.
* Order + OrderStatus models and seed script.
* Payment integration: create-payment flow using gateway create-collect-request and signed `sign` JWT.
* Webhook endpoint to receive gateway callbacks and update order status.
* Transactions endpoints: list, by-school, by-custom-order-id.

# 3. Key File Structure

**Important folders and files (backend/):**

* server.js
* app.js
* config/db.js
* routes/auth.routes.js, payment.routes.js, webhook.routes.js, transactions.routes.js
* controllers/auth.controller.js, payment.controller.js, webhook.controller.js, transactions.controller.js
* models/User.js, Order.js, OrderStatus.js, WebhookLog.js
* middlewares/auth.middleware.js, error.middleware.js
* services/payment.service.js, jwt.service.js
* seed/seed.js, .env.example, README.md

# 4. .env (sample)

Create backend/.env from .env.example. Do NOT commit this file. Replace placeholders with real values (do not share secrets).

PORT=5000  
MONGO\_URI=mongodb://localhost:27017/edviron  
JWT\_SECRET=<your\_generated\_jwt\_secret\_here>  
JWT\_EXPIRES\_IN=7d  
  
PAYMENT\_BASE\_URL=https://dev-vanilla.edviron.com/erp  
PAYMENT\_API\_KEY=<provider\_api\_key\_here>  
PAYMENT\_PG\_KEY=edvtest01  
SCHOOL\_ID=65b0e6293e9f76a9694d84b4  
  
NODE\_ENV=development  
FRONTEND\_BASE\_URL=http://localhost:3000

# 5. Payment service (key snippet)

This service generates the `sign` JWT (HS256) using PAYMENT\_PG\_KEY and calls the gateway endpoint with Authorization header using PAYMENT\_API\_KEY.

// backend/services/payment.service.js (core)  
const jwt = require('jsonwebtoken');  
const axios = require('axios');  
  
const signPayload = { school\_id: String(SCHOOL\_ID), amount: String(amount), callback\_url: String(callback\_url) };  
const sign = jwt.sign(signPayload, process.env.PAYMENT\_PG\_KEY, { algorithm: 'HS256' });  
  
const body = { school\_id: signPayload.school\_id, amount: signPayload.amount, callback\_url: signPayload.callback\_url, sign };  
const headers = { 'Content-Type': 'application/json', 'Authorization': `Bearer ${process.env.PAYMENT\_API\_KEY}` };  
const resp = await axios.post(`${process.env.PAYMENT\_BASE\_URL}/create-collect-request`, body, { headers });

# 6. create-payment controller (core)

// backend/controllers/payment.controller.js (core)  
const order = await Order.create({ school\_id: process.env.SCHOOL\_ID, student\_info, custom\_order\_id });  
const orderStatus = await OrderStatus.create({ collect\_id: order.\_id, order\_amount, status: 'initiated' });  
const gatewayResp = await callCreateCollectRequest({ amount: order\_amount, callback\_url: cb });  
order.collect\_request\_id = gatewayResp.collect\_request\_id;  
order.collect\_request\_url = gatewayResp.Collect\_request\_url;  
await order.save();  
return res.json({ order\_id: order.\_id, collect\_request\_id: order.collect\_request\_id, payment\_url: order.collect\_request\_url, gateway\_response: gatewayResp });

# 7. Webhook (sample payload)

POST /api/webhook  
{  
 "order\_info": {  
 "order\_id":"<collect\_request\_id>",  
 "order\_amount":2000,  
 "transaction\_amount":2000,  
 "payment\_mode":"card",  
 "bank\_reference":"BANK-123",  
 "status":"success",  
 "payment\_time":"2025-01-01T12:00:00Z"  
 }  
}

# 8. Test Endpoints (Postman / curl)

Run these in order. Replace placeholders with real values.

- Health check: GET / -> http://localhost:5000/

- Register: POST /api/auth/register -> body: {email,password,name}

- Login: POST /api/auth/login -> body: {email,password} (saves token)

- List transactions: GET /api/transactions (Auth header: Bearer <token>)

- Create payment: POST /api/payment/create-payment (Auth) -> body: {student\_info, order\_amount, callback\_url}

- Check status (backend): GET /api/payment/status/:collect\_request\_id (Auth)

- Webhook (simulate): POST /api/webhook -> sample payload (see above)

- Transactions by school: GET /api/transactions/school/:schoolId (Auth)

- Transaction by custom id: GET /api/transaction-status/:custom\_order\_id (Auth)

# 9. Testing steps (quick)

1. Start MongoDB (local or Atlas). 2. npm install && npm run seed. 3. npm run dev. 4. Use Postman in this order: login -> create-payment -> open payment\_url -> simulate payment -> check status or webhook. 5. Verify DB collections: orders, orderstatuses, webhooklogs.

# 10. Remaining / Recommended Tasks

* Add input validation (express-validator or Joi) for all endpoints.
* Implement webhook signature verification if gateway provides it.
* Add unit/integration tests (jest + supertest).
* Set up logging (winston/pino) and add monitoring/health checks.
* Lock down CORS for production and secure secrets with secret manager.

# Notes

This document excludes sensitive keys. Replace placeholders with your actual .env values. Keep secrets out of source control.