Backend

Backend documentation

Introduction

Introduction

Overview

OnDelivery API (OSAS - OnDelivery Smart Agent System) is a comprehensive multi-courier shipping logistics platform built with Node.js. The system enables:

- **Multi-courier aggregation** - Integrate and manage multiple shipping partners (JNE, Lion Parcel, JT Cargo, AnteRaja, etc.)
- **Real-time package tracking** - Track shipments from pickup to delivery
- **Dynamic pricing** - Calculate shipping fees based on origin, destination, weight, and service type
- **Warehouse operations** - Manage scanning, manifesting, handovers, and quality control
- **Financial management** - Handle billing, commissions, and VIP customer pricing
- **Driver mobile app** - Support delivery drivers with marketplace integration
- **Marketplace integration** - Connect with multiple e-commerce platforms

Project Name: ondelivery-api
Version: 1.0.0
Main Entry Point: `ondeliv-backend.js`
Default Port: 4220
License: ISC

Architecture & Technology Stack

Core Technologies

Technology	Version	Purpose
Node.js	-	Runtime environment
Express.js	5.2.1	Web application framework
PostgreSQL	-	Primary database (dual setup: OSAS + ONMART)
Sequelize	6.37.7	ORM for database operations

Key Dependencies

Authentication & Security:

- `jsonwebtoken` (9.0.2) - JWT token generation and verification
- `passport` (0.7.0) - Authentication middleware
- `passport-http` (0.3.0) - HTTP Basic authentication
- `bcryptjs` (3.0.3) - Password hashing
- `helmet` (8.1.0) - Security headers
- `express-rate-limit` (8.2.1) - Rate limiting protection

Data Validation & Processing:

- `ajv` (8.17.1) - JSON schema validation
- `ajv-formats` (3.0.1) - Additional format validators
- `ajv-keywords` (5.1.0) - Extended validation keywords

HTTP & External APIs:

- `axios` (1.13.2) - HTTP client for external API calls
- `cors` (2.8.5) - Cross-origin resource sharing

File Processing:

- `multer` (2.0.2) - File upload handling
- `sharp` (0.34.5) - Image processing and optimization
- `exceljs` (4.4.0) - Excel file generation
- `pdfmake` (0.2.20) - PDF document generation
- `xlsx` (0.20.3) - Excel file parsing

Utilities:

- `moment` (2.30.1) / `dayjs` (1.11.19) - Date/time manipulation
- `lodash` (4.17.21) - Utility functions
- `node-cron` (4.2.1) - Scheduled tasks
- `nodemailer` (7.0.11) - Email sending
- `nanoid` (5.1.6) - Unique ID generation
- `morgan` (1.10.1) - HTTP request logging
- `cli-color` (2.0.4) - Colorized console output

Getting Started

Prerequisites

- **Node.js** (v14+ recommended)
- **PostgreSQL** (v15+ recommended)
- **npm** or **yarn** package manager

Installation

```
# Clone the repository
git clone <repository-url>
cd ondelivery-api

# Install dependencies
npm install

# Setup environment variables
# Copy .env.example to .env and configure
cp .env.example .env
```

Configuration

1. **Database Setup:**
   - Create two PostgreSQL databases (OSAS and ONMART)
   - Update database credentials in `.env`
2. **Environment Variables:**
   - Configure all required environment variables (see Environment Variables)
3. **Initialize Database:**

   ```
   # Run migrations if available
   # The application uses Sequelize ORM
   ```

Running the Application

```
# Development mode with auto-reload
npm start

# Production mode
node ondeliv-backend.js
```

The server will start on port **4220** (or as configured in `PORT` environment variable).

Verify Installation

Visit `http://localhost:4220/` - you should see:

```
{
  "message": "This backend osas works well for now"
}
```

Core Features

1. Waybill Management

- **Create Waybills** - Generate shipping labels with unique AWB (Air Waybill) numbers
- **Edit Waybills** - Modify shipment details before dispatch
- **Track Waybills** - Real-time tracking with status updates
- **Multi-AWB Operations** - Batch operations on multiple waybills

2. Scanning Operations

Outgoing Scans:

- **Packing** - Package preparation and verification
- **Handover** - Transfer between logistics hubs

Incoming Scans:

- **Receiving** - Package arrival at distribution center
- **Delivery** - Final delivery to customer
- **Problem Scan** - Exception handling (damaged, lost, etc.)

3. Courier Partner Integration

- Support for 10+ courier partners
- Webhook-based status synchronization
- API-based shipment creation
- Automatic status mapping to internal codes

4. Fee Calculation

- Dynamic pricing based on:
  - Origin and destination locations
  - Package weight and dimensions
  - Service type (regular, express, cargo)
  - Customer-specific pricing (VIP)

    - Special location surcharges

5. Reports & Analytics

Operational Reports:

- Monitoring pickup/delivery status
- Manifest reports
- Arrival/departure reports
- SLA performance tracking

Financial Reports:

- Billing reports (standard & VIP)
- Agent commission calculations
- Distribution center (DC) commissions
- Finance reconciliation

Quality Reports:

- Courier performance
- Problem scan analysis
- Weight discrepancy tracking

6. Quality Management

- **Weight Verification** - Alter weight for discrepancies
- **POD (Proof of Delivery)** - Capture delivery signatures/photos
- **POP (Proof of Pickup)** - Document package pickup with photos

7. Driver App Support

- Mobile-friendly endpoints for delivery drivers
- Marketplace order management
- Route optimization support
- Real-time status updates

8. Marketplace Integration

- **Pickup Management** - Handle marketplace order pickups
- **Receiving** - Process incoming marketplace orders

- Support for multiple platforms

9. Administration

- **Account Management** - User and agent administration
- **Customer Management** - Customer profiles and preferences
- **Feed Information** - System notifications and updates

10. Financial Operations

- **VIP Billing** - Custom pricing for premium customers
- **Payment Gateway** - Xendit integration for payments
- **Commission Tracking** - Calculate agent and DC commissions

---

Directory Structure

```
ondelivery-api/
├── ondeliv-backend.js          # Main application entry point
├── package.json                # Dependencies and scripts
├── README.md                   # Basic project information
├── BACKEND_DOCUMENTATION.md    # This file
│
├── app/                        # Application code
│   ├── config/                 # Configuration files
│   │   ├── api.js              # External API endpoints
│   │   ├── auth.config.js      # JWT secret configuration
│   │   └── db.config.js        # Database connection config
│   │
│   ├── controllers/            # Business logic controllers
│   │   ├── 3p/                 # 3rd-party courier integrations
│   │   │   ├── jne.controller.js
│   │   │   ├── lion-parcel.controller.js
│   │   │   ├── jt-cargo.controller.js
│   │   │   ├── anteraja.controller.js
│   │   │   └── sapx.controller.js
│   │   ├── auth/               # Authentication controllers
│   │   ├── check-fee/          # Fee calculation logic
│   │   ├── driver-app/         # Driver mobile app
│   │   ├── finance/            # Financial operations
│   │   ├── incoming-scans/     # Receiving, delivery, problems
│   │   ├── marketplace/        # Marketplace integrations
│   │   ├── outgoing-scans/     # Packing, handover operations
│   │   ├── quality-management/ # Weight, POD, POP
│   │   ├── reports/            # Report generation
│   │   ├── trucking/           # Trucking operations
│   │   └── waybill/            # Waybill CRUD operations
│   │
│   ├── middleware/             # Express middleware
│   │   ├── authJwt.js          # JWT token verification
│   │   ├── passport.js         # Passport strategies
```

```
│   │   └── error_param_handler.js # Error handling
│   │
│   ├── models/               # Sequelize ORM models (52+ models)
│   │   ├── index.js          # Model aggregation
│   │   ├── scans.model.js    # Tracking status records
│   │   ├── partners.model.js # Courier partner data
│   │   ├── user.model.js     # User accounts
│   │   ├── waybill_numbers.model.js # AWB tracking
│   │   ├── resi_3p.model.js  # 3P waybill mapping
│   │   └── [48+ other models]
│   │
│   ├── routes/               # API route definitions
│   │   ├── auth.routes.js    # Authentication endpoints
│   │   ├── waybill/          # Waybill routes
│   │   ├── incoming_scans/   # Incoming scan routes
│   │   ├── outgoing_scans/   # Outgoing scan routes
│   │   ├── reports/          # Report routes
│   │   ├── 3p.routes.js      # 3P integration webhooks
│   │   └── [35+ route files]
│   │
│   ├── schemas/              # JSON validation schemas (AJV)
│   │   └── [validation definitions]
│   │
│   ├── schedulers/           # Cron job definitions
│   │   ├── update-resi-status.js
│   │   └── transaction-terminator.js
│   │
│   ├── status/               # Status mapping JSON files
│   │   └── [courier status mappings]
│   │
│   ├── utility/              # Helper utilities
│   │   ├── check-fee.util.js # Fee calculation helpers
│   │   ├── service.util.js   # Service utilities
│   │   ├── waybill.util.js   # Waybill helpers
│   │   └── web-hook.js       # Webhook utilities
│   │
│   ├── maping/               # Data mapping files
│   │   └── jntcargo-special-pricing.json
│   │
│   └── fonts/                # Custom fonts for PDF generation
```

```
│
├── assets/                  # Static assets
├── resources/               # Resource files
│   └── temp/                # Temporary files
│       ├── img/             # Image uploads
│       │   ├── pickup/
│       │   ├── delivery/
│       │   ├── pickup_eatsok/
│       │   ├── delivery_eatsok/
│       │   └── agent/
│       └── [generated files]
│
├── scripts/                 # Utility scripts
├── tests/                   # Test files
├── check_fee_test.js        # Fee calculation tests
├── check_price.js           # Price checking script
└── debug_provinces.js       # Province debugging utility
```

Database Architecture

The application uses **two PostgreSQL databases**:

1. **OSAS Database** (Primary)
   - Main operational database
   - Contains waybills, scans, users, partners, etc.
2. **ONMART Database** (Secondary)
   - Marketplace-specific operations
   - Separate schema for marketplace data

Key Database Models

Core Operational Models

Model	Purpose	Key Fields
`scans`	Tracking status history	waybill_id, status, location, timestamp
`waybill_numbers`	Shipment records	awb_number, origin, destination, weight
`partners`	3P courier partners	name, api_token, webhook_url
`resi_3p`	3P waybill mapping	internal_awb, partner_awb, partner_id
`user`	User accounts	username, password_hash, role_id, agent_id

Scanning Models

Model	Purpose
`delivery_scan`	Delivery completion records
`handover_scan`	Hub-to-hub transfers
`receiving_scan`	Package arrivals
`problem_scan`	Exception tracking
`packing_scan`	Package preparation

Financial Models

Model	Purpose
billing_vip_config	VIP customer pricing
xendit_payment	Payment gateway records
xendit_payout	Payout transactions
agent_commission	Commission calculations

Master Data Models

Model	Purpose
agent_locations	Agent location mapping
agent_customer	Customer-agent relationships
awb_books	AWB number allocation
courier_partners	Courier configurations
service_types	Available services

Additional Models (52 total)

- Manifest management (manifest_pcp, manifest_pos)
- Quality management (alter_weight, pod_record, pop_record)
- Trucking operations (trucking_shipment, trucking_config)
- Location master data (all_location_migration)
- Marketplace integration models
- Report configuration models

Database Relationships

```
User ──┐
       ├─→ Agent_Locations → Waybill_Numbers
       └─→ Role

Waybill_Numbers ──┬─→ Scans (1:many)
                  ├─→ Resi_3p
                  └─→ Delivery_Scan / Problem_Scan

Partners ─→ Resi_3p ─→ Waybill_Numbers

Agent_Customer ─→ Waybill_Numbers
```

Connection Pooling

```
// From db.config.js
pool: {
  max: DB_POOL_MAX,        // Maximum connections
  min: DB_POOL_MIN,        // Minimum connections
  acquire: DB_POOL_ACQUIRE, // Connection acquisition timeout
  idle: DB_POOL_IDLE       // Idle timeout
}
```

Authentication and security

Authentication Methods

1. JWT Token Authentication (Primary)

Usage: Protected API endpoints for authenticated users

Implementation:

- Middleware: `authJwt.verifyToken`
- Header: `Authorization: Bearer <token>`
- Token includes: `id`, `agentId`, `username`, `role`, `location_id`, `name`

Example:

```
// Route protection
router.post('/api/waybill/create',
  [authJwt.verifyToken],
  controller.createWaybill
);
```

Token Payload:

```
{
  id: 123,
  agentId: 456,
  username: "john.doe",
  role: "admin",
  location_id: 789,
  name: "John Doe"
}
```

2. Passport HTTP Basic Authentication

Usage: 3rd-party API integrations (webhooks, external calls)

Strategies:

- `eatsok` - EatSOK food delivery partner
- `anteraja` - AnteRaja courier integration
- `linked` - Linked account authentication
- `integrationproc` - Integration processor

Implementation:

```
// Basic Auth header: Authorization: Basic base64(username:password)
passport.authenticate('eatsok', { session: false })
```

Credentials Validated Against:

- `EATSOK_SECRET_KEY_ONDEL`
- `ANTERAJA_SECRET_KEY_ONDEL`
- `LINKED_ACCOUNT_SECRET_KEY`
- `INTEGRATION_PROC_KEY`

Security Features

Rate Limiting

Configuration:

```
windowMs: RATE_LIMIT_WINDOW_MS || 10000,  // 10 seconds
max: RATE_LIMIT_MAX || 20,                // 20 requests per window
message: "You are sending requests too quickly. Please wait 1 second."
```

Applied to: All routes globally

CORS Protection

```
cors({
  origin: ORIGIN_CORS.split(','),  // Whitelist from env
  allowedHeaders: ['Content-Type', 'Authorization']
})
```

Helmet Security Headers

- XSS protection
- Content Security Policy
- DNS prefetch control
- Frame options
- HSTS (HTTP Strict Transport Security)

Additional Security

Password Security:

- Hashing: `bcryptjs` with salt rounds
- No plaintext password storage

Input Validation:

- JSON Schema validation via AJV
- Request parameter sanitization
- File upload size limits (10MB)

Permission Headers:

```
Permissions-Policy:
  geolocation=(self),
  camera=(),
  microphone=(),
  fullscreen=(self),
  payment=(self)
```

File Access Control:

```
Cross-Origin-Resource-Policy: cross-origin
```

Role-Based Access Control (RBAC)

Role Model:

- Stored in `role` table
- Associated with `user` via `role_id`

Access Menu Control:

- `accessmenu` model defines permissions
- Location-based filtering via `agent_locations`

Common Roles:

- Admin
- Agent
- Driver
- Warehouse Staff

- Finance

API Endpoints

Authentication Endpoints

Method	Endpoint	Auth	Description
POST	`/api/auth/signup`	None	Register new user
POST	`/api/auth/signin`	None	Login and get JWT token
GET	`/api/auth/profile`	JWT	Get user profile
GET	`/api/auth/agents`	JWT	List all agents

Waybill Endpoints

Method	Endpoint	Auth	Description
POST	`/api/waybill/create`	JWT	Create new waybill
GET	`/api/waybill/:id`	JWT	Get waybill details
PUT	`/api/waybill/:id`	JWT	Update waybill
DELETE	`/api/waybill/:id`	JWT	Cancel waybill
GET	`/api/waybill/track/:awb`	JWT	Track waybill by AWB
POST	`/api/waybill/bulk`	JWT	Bulk waybill creation
GET	`/api/waybill/view-img/:id`	JWT	View waybill image

Check Fee Endpoints

Method	Endpoint	Auth	Description
POST	`/api/check-fee`	JWT	Calculate shipping fee
POST	`/api/check-fee/bulk`	JWT	Bulk fee calculation
GET	`/api/check-fee/services`	JWT	Get available services

Scanning Endpoints

Outgoing Scans:

Method	Endpoint	Auth	Description
POST	`/api/scan/packing`	JWT	Record packing scan
POST	`/api/scan/handover`	JWT	Record handover scan
GET	`/api/scan/handover/:id`	JWT	Get handover details

Incoming Scans:

Method	Endpoint	Auth	Description
POST	`/api/scan/receiving`	JWT	Record receiving scan
POST	`/api/scan/delivery`	JWT	Record delivery scan
POST	`/api/scan/problem`	JWT	Record problem scan
GET	`/api/scan/delivery/:id`	JWT	Get delivery details

Courier Partner Endpoints

Method	Endpoint	Auth	Description
GET	`/api/partners`	JWT	List all partners
GET	`/api/partners/:id`	JWT	Get partner details
POST	`/api/partners`	JWT	Add new partner
PUT	`/api/partners/:id`	JWT	Update partner
DELETE	`/api/partners/:id`	JWT	Remove partner

3rd-Party Integration Endpoints

Method	Endpoint	Auth	Description
POST	`/api/3p/jne/webhook`	Basic	JNE status webhook
POST	`/api/3p/lionparcel/webhook`	Basic	Lion Parcel webhook
POST	`/api/3p/jtcargo/webhook`	Basic	JT Cargo webhook
POST	`/api/3p/anteraja/webhook`	Basic	AnteRaja webhook
POST	`/api/3p/sicepat/webhook`	Basic	Sicepat webhook

Reports Endpoints

Method	Endpoint	Auth	Description
GET	`/api/reports/monitoring-pickup`	JWT	Pickup monitoring report
GET	`/api/reports/monitoring-delivery`	JWT	Delivery monitoring report
GET	`/api/reports/manifest`	JWT	Manifest report
GET	`/api/reports/sla`	JWT	SLA performance report
GET	`/api/reports/billing`	JWT	Billing report
GET	`/api/reports/billing-vip`	JWT	VIP billing report
GET	`/api/reports/commission`	JWT	Commission report
GET	`/api/reports/finance`	JWT	Finance report
GET	`/api/reports/courier-performance`	JWT	Courier performance

Quality Management Endpoints

Method	Endpoint	Auth	Description
POST	`/api/qm/alter-weight`	JWT	Record weight alteration
POST	`/api/qm/pod`	JWT	Upload proof of delivery
POST	`/api/qm/pop`	JWT	Upload proof of pickup
GET	`/api/qm/pod/:awb`	JWT	Get POD for waybill
GET	`/api/qm/pop/:awb`	JWT	Get POP for waybill

Driver App Endpoints

Method	Endpoint	Auth	Description
GET	`/api/driver/tasks`	JWT	Get assigned tasks
POST	`/api/driver/pickup`	JWT	Complete pickup
POST	`/api/driver/delivery`	JWT	Complete delivery
GET	`/api/driver/marketplace/orders`	JWT	Get marketplace orders

Marketplace Endpoints

Method	Endpoint	Auth	Description
POST	`/api/marketplace/pickup`	JWT	Record marketplace pickup

Method	Endpoint	Auth	Description
POST	`/api/marketplace/receiving`	JWT	Record marketplace receiving
GET	`/api/marketplace/orders`	JWT	Get marketplace orders

Administration Endpoints

Method	Endpoint	Auth	Description
GET	`/api/admin/users`	JWT	List users
POST	`/api/admin/users`	JWT	Create user
PUT	`/api/admin/users/:id`	JWT	Update user
DELETE	`/api/admin/users/:id`	JWT	Delete user
GET	`/api/admin/customers`	JWT	List customers
POST	`/api/admin/feed-info`	JWT	Post system notification

Finance Endpoints

Method	Endpoint	Auth	Description
GET	`/api/finance/billing-vip`	JWT	Get VIP billing configs
POST	`/api/finance/billing-vip`	JWT	Create VIP billing config
PUT	`/api/finance/billing-vip/:id`	JWT	Update VIP billing config

File Download

Method	Endpoint	Auth	Description
GET	`/api/file/download/:filename`	None	Download generated file

Note: File is automatically deleted after download.

Courier Partner Integrations

Supported Courier Partners

The system integrates with multiple courier partners in Indonesia:

1. JNE (Jalur Nugraha Ekakurir)

Features:

- Status webhook integration
- Status code mapping to internal codes
- Real-time tracking updates

Controller: `app/controllers/3p/jne.controller.js`

2. Lion Parcel

Features:

- Scheduled pickup requests via cron job
- Bearer token authentication
- Automatic pickup scheduling

Configuration:

```
// Environment variable
LIONPARCEL_PICKUP_CRON="*/30 * * * *"  // Every 30 minutes
LIONPARCEL_BEARER_TOKEN="<token>"
```

Controller: `app/controllers/3p/lion-parcel.controller.js`

3. JT Cargo (J&T Cargo)

Features:

- Full API integration
- Special pricing support
- Database-driven pricing lookup
- Fallback to JSON mapping

Special Pricing:

- Primary: `jnt_cargo_special_prices` table
- Fallback: `app/maping/jntcargo-special-pricing.json`

Controller: `app/controllers/3p/jt-cargo.controller.js`

4. AnterAja

Features:

- Webhook integration
- Access key authentication
- Status synchronization

Authentication:

- Access Key ID: `ANTERAJA_ACCESS_KEY_ID`
- Secret Access Key: `ANTERAJA_SECRET_ACCESS_KEY`

Controller: `app/controllers/3p/anteraja.controller.js`

5. Sicepat

Features:

- Status webhook support
- Tracking integration

Controller: `app/controllers/3p/sicepat.controller.js`

6. Sap Express (SAPX)

Features:

- API integration
- Shipment creation

Controller: `app/controllers/3p/sapx.controller.js`

7. POS Indonesia

Features:

- Indonesian postal service integration
- Manifest generation
- Token-based authentication

Routes: `app/routes/pos_indo.routes.js`

8. PCP Express

Features:

- PCP carrier network
- Manifest management

Routes: `app/routes/pcp_express.routes.js`

9. Laris Cargo

Features:

- Regional courier integration

Routes: `app/routes/laris_cargo.routes.js`

Integration Pattern

All courier integrations follow a similar pattern:

```
// 1. Receive webhook from partner
POST /api/3p/{partner}/webhook

// 2. Validate authentication (Basic Auth or API Key)
passport.authenticate('{partner}Strategy')

// 3. Parse partner status code
parsePartnerStatus(partnerStatusCode)

// 4. Map to internal status code
const internalStatus = statusMapping[partnerStatusCode]

// 5. Update internal tracking
await updateScan(awb, internalStatus)

// 6. Trigger internal webhooks (if configured)
await triggerWebhook(waybillData)

// 7. Return acknowledgment
```

```
return { success: true }
```

Status Mapping

Each courier has status mappings in `app/status/` directory:

- Maps partner-specific status codes to internal OnDelivery codes
- Ensures consistent tracking across all partners
- Stored as JSON configuration files

Example Status Flow:

```
Partner: "DELIVERED"
  → Internal: "DLV"
  → Display: "Delivered to Customer"
```

Resi 3P Mapping

Purpose: Link internal AWB numbers to partner AWB numbers

Table: `resi_3p`

Fields:

- `internal_awb` - OnDelivery waybill number
- `partner_awb` - Partner's tracking number
- `partner_id` - Courier partner ID
- `status` - Current status
- `created_at` - Mapping creation time

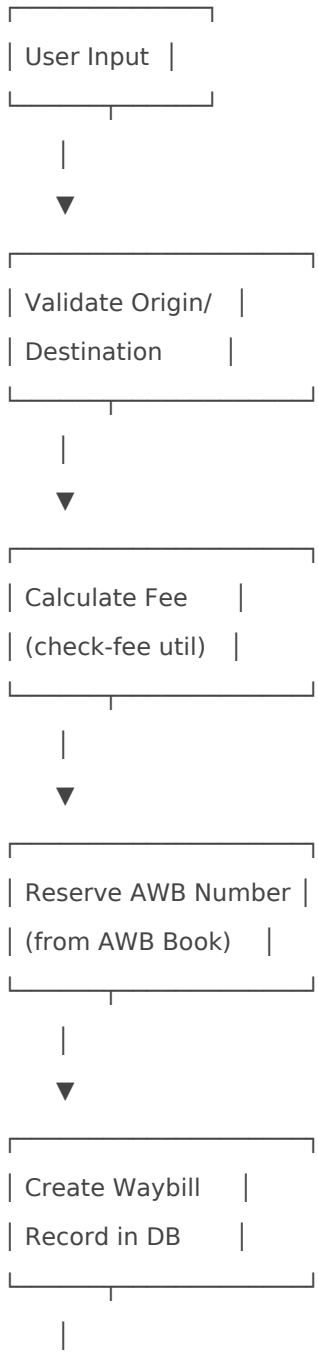
Usage:

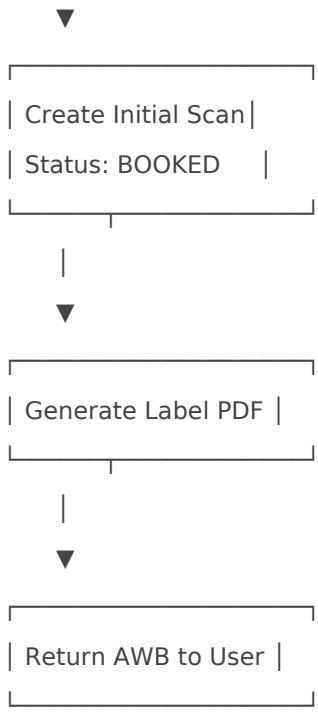
```
// When creating shipment with partner
const mapping = await Resi3P.create({
  internal_awb: "ONX1234567890",
  partner_awb: "JNE0987654321",
  partner_id: 1  // JNE
});

// When receiving webhook
const waybill = await findWaybillByPartnerAwb(
```

```
  webhookData.awb,
  partnerConfig.id
);
```

Business Workflows

Workflow 1: Waybill Creation & Booking





Workflow 2: Package Tracking Lifecycle

Outgoing Phase:

```
BOOKED → PACKED → HANDOVER → DEPARTURE
```

Transit Phase:

```
DEPARTURE → [TRANSIT] → ARRIVAL
```

Delivery Phase:

```
ARRIVAL → RECEIVING → OUT_FOR_DELIVERY → DELIVERED
                                       → PROBLEM
```

Exception Handling:

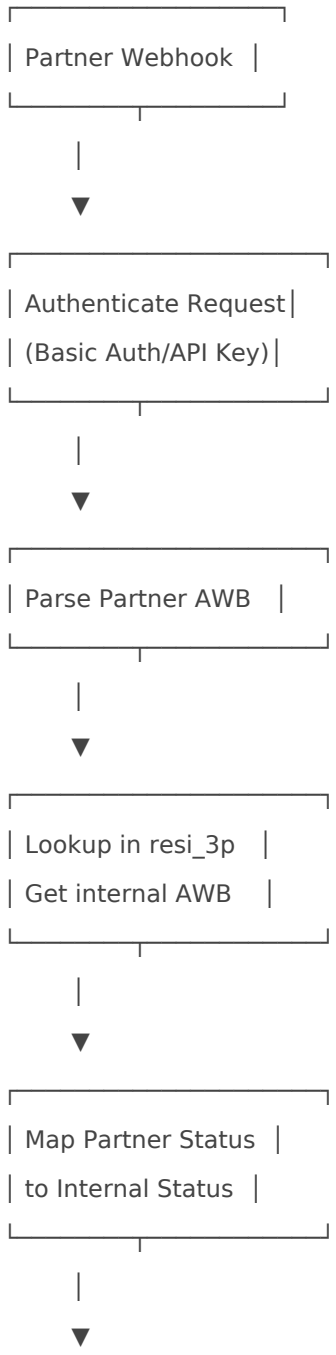
```
ANY_STATUS → PROBLEM → [PROBLEM_RESOLVED]
           → RETURN → [RETURN_COMPLETE]
           → CANCEL
```

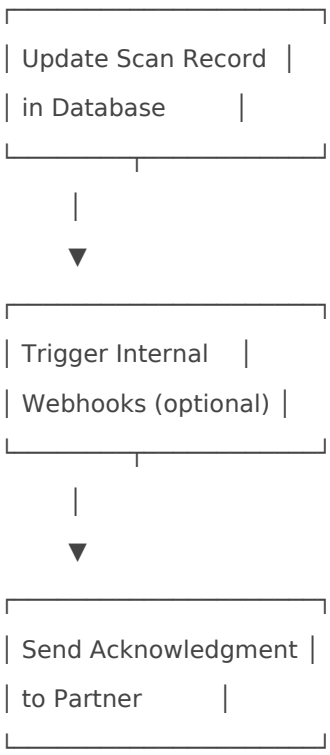
Status Codes:

- `BKD` - Booked
- `PKD` - Packed

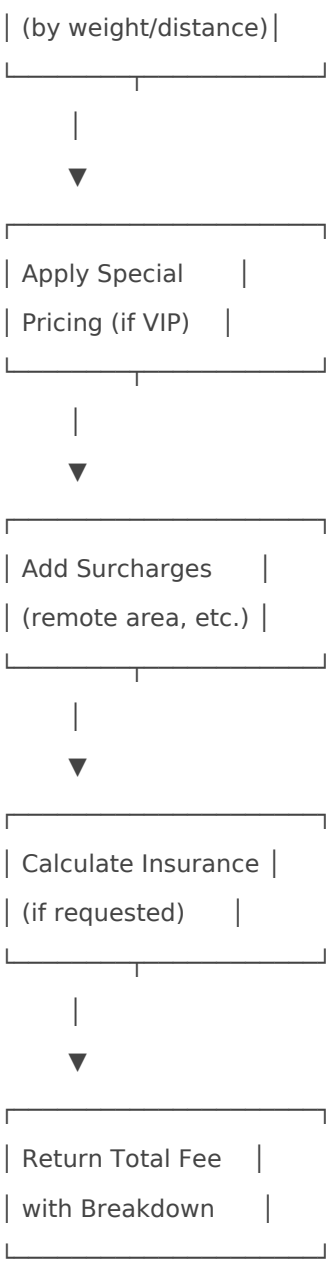
- `HND` - Handover
- `DPT` - Departure
- `ARV` - Arrival
- `RCV` - Receiving
- `OFD` - Out for Delivery
- `DLV` - Delivered
- `PRB` - Problem
- `RTN` - Return
- `CNL` - Cancel

Workflow 3: 3rd-Party Integration





Workflow 4: Fee Calculation



Workflow 5: Scanning Operations

Packing Scan:

1. Scan AWB number
2. Verify waybill exists
3. Check current status (must be BOOKED)
4. Record packing details (weight, dimensions)
5. Update status to PACKED
6. Generate packing list

Handover Scan:

1. Create handover manifest
2. Scan multiple AWBs
3. Verify all AWBs are PACKED
4. Record handover details (from/to location)
5. Update all AWBs to HANDOVER status
6. Generate handover manifest PDF
7. Print manifest for courier

Receiving Scan:

1. Scan incoming AWBs
2. Match with expected manifest
3. Record arrival time
4. Update status to RECEIVING
5. Flag any discrepancies
6. Assign to delivery driver

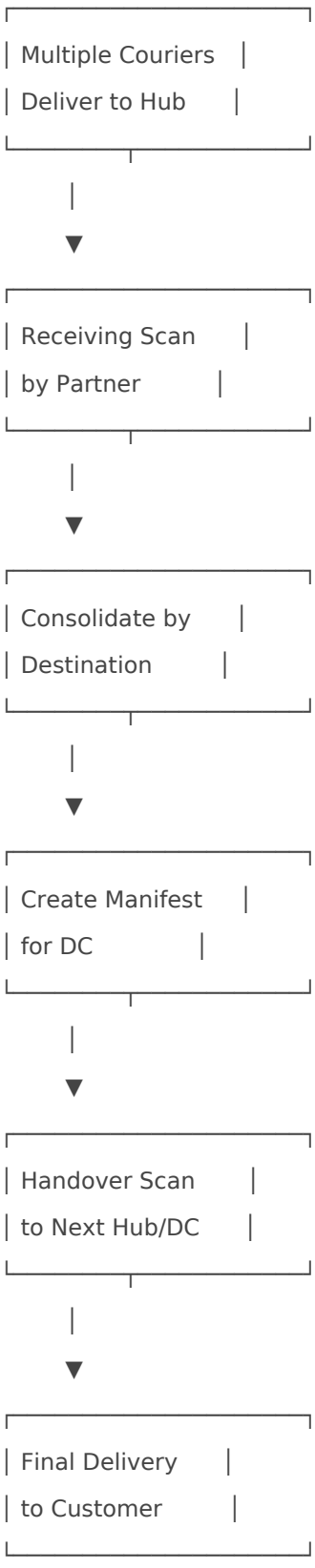
Delivery Scan:

1. Driver scans AWB
2. Capture POD (photo/signature)
3. Record delivery details
4. Update status to DELIVERED
5. Send customer notification
6. Update commission records

Problem Scan:

1. Scan AWB with issue
2. Select problem type (damaged, lost, refused, etc.)
3. Capture photos
4. Add notes
5. Update status to PROBLEM
6. Trigger problem resolution workflow
7. Notify relevant parties

Workflow 6: Multi-Courier Consolidation



Workflow 7: Quality Management

Weight Discrepancy Handling:

1. Package weighed at receiving
2. Compare with declared weight
3. If difference > threshold:
   - Capture photo evidence
   - Record actual weight
   - Create alter_weight record
   - Calculate price adjustment
   - Notify customer
4. Update billing if needed

Proof of Delivery (POD):

1. Driver completes delivery
2. Capture recipient signature (digital)
3. Take delivery photo
4. Record GPS coordinates
5. Upload to server
6. Link to waybill
7. Status → DELIVERED

Proof of Pickup (POP):

1. Driver arrives at pickup location
2. Take package photos
3. Capture sender signature
4. Record GPS coordinates
5. Upload to server
6. Link to waybill
7. Status → BOOKED

External API Integrations

Configured External APIs

1. Agent Management System

Purpose: Manage agent locations and configurations

Configuration:

- Base URL: `API_URL_AGENT_ONAGENT`
- Authentication: JWT token
- Usage: Agent CRUD operations

2. User State Management

Purpose: Track user session and state

Configuration:

- Base URL: `API_URL_USERSTATE`
- Authentication: JWT token
- Usage: User activity tracking

3. User OnDelivery System

Purpose: User profile management

Configuration:

- Base URL: `API_URL_USERONDELIV`
- Authentication: JWT token
- Usage: User profile operations

4. Sunshine Location Service

Purpose: Location and area data

Configuration:

- Base URL: `API_URL_SUNSHINE`
- Authentication: API key
- Usage: Location lookup, province/city data

5. APISAT

Purpose: Satellite tracking (purpose unclear from code)

Configuration:

- Base URL: `API_URL_APISAT` and `APISAT_LINK`
- Authentication: API key
- Usage: External tracking integration

6. EatSOK

Purpose: Food delivery partner integration

Configuration:

- Base URL: `API_EATSOK`
- Authentication: Basic Auth
  - API Key: `EATSOK_API_KEY`
  - Secret: `EATSOK_SECRET_KEY_ONDEL`
- Special Features:
  - Separate image directories
  - Custom scanning workflow

7. Xendit Payment Gateway

Purpose: Payment processing and payouts

Features:

- Payment collection
- Payout disbursement
- Transaction tracking

Models:

- `xendit_payment` - Payment records
- `xendit_payout` - Payout records

8. Blibli Marketplace

Purpose: E-commerce integration

Configuration:

- Webhook Key: `BLIBLI_WEBHOOK_KEY`
- Authentication: Webhook signature validation
- Features: Order notifications, fulfillment

9. OnMarket Platform

Purpose: Marketplace platform integration

Configuration:

- Webhook Key: `ONMARKET_WEBHOOK_KEY`
- Authentication: Webhook signature validation
- Features: Multi-marketplace aggregation

API Utility Modules

Location: `app/config/api.js`

Purpose: Centralized API endpoint configuration

Usage:

```
const api = require('./config/api');


// Make external API call
const response = await axios.get(api.AGENT_SYSTEM + '/agents');
```

Webhook Handling

Incoming Webhooks:

1. **Courier Partner Webhooks**
   - Endpoint: `/api/3p/{partner}/webhook`
   - Authentication: Basic Auth or API Key
   - Purpose: Status updates from couriers
2. **Marketplace Webhooks**
   - Endpoint: `/api/marketplace/webhook`
   - Authentication: Webhook signature
   - Purpose: New order notifications

Outgoing Webhooks:

Utility: `app/utility/web-hook.js`

Purpose: Notify external systems of status changes

Configuration:

- Configured per partner/customer
- Triggered on status updates
- Includes retry logic

Example:

```
// Trigger webhook on status change
await triggerWebhook({
  awb: "ONX1234567890",
  status: "DELIVERED",
  timestamp: new Date(),
  location: "Jakarta"
});
```

Environment Variables

Required Environment Variables

Database Configuration

```
# PostgreSQL - OSAS Database
DB_HOST=localhost
DB_PORT=5432
DB_USER=postgres
DB_PASSWORD=your_password
DBS=osas_db
DB_DIALECT=postgres

# Connection Pool
DB_POOL_MAX=5
DB_POOL_MIN=0
DB_POOL_ACQUIRE=30000
DB_POOL_IDLE=10000

# ONMART Database
ONMART_DB_HOST=localhost
ONMART_DB_PORT=5432
ONMART_DB_USER=postgres
ONMART_DB_PASSWORD=your_password
ONMART_DBS=onmart_db
```

Application Configuration

```
# Server
PORT=4220
BASEURL=http://localhost:4220

# JWT Authentication
JWT_SECRET=your_jwt_secret_key
```

```
JWT_EXPIRATION=86400  # 24 hours in seconds

# Custom Configuration
CUSTOM_ALPHABET=ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
```

Security Configuration

```
# Rate Limiting
RATE_LIMIT_WINDOW_MS=10000  # 10 seconds
RATE_LIMIT_MAX=20           # 20 requests per window

# CORS
ORIGIN_CORS=http://localhost:3000,https://yourdomain.com
```

External API URLs

```
# Internal Services
API_URL_AGENT_ONAGENT=http://agent-system/api
API_URL_USERONDELIV=http://user-service/api
API_URL_USERSTATE=http://user-state/api
API_URL_SUNSHINE=http://sunshine-api/api
API_URL_APISAT=http://apisat/api
APISAT_LINK=http://apisat-link/api

# Partner APIs
API_EATSOK=https://eatsok-api.com
```

Courier Partner Credentials

```
# JNE
JNE_API_KEY=your_jne_api_key
JNE_USERNAME=your_jne_username
JNE_PASSWORD=your_jne_password

# Lion Parcel
LIONPARCEL_BEARER_TOKEN=your_lion_parcel_token
LIONPARCEL_PICKUP_CRON=*/30 * * * *  # Every 30 minutes

# AnteRaja
ANTERAJA_URL=https://anteraja-api.com
```

```
ANTERAJA_ACCESS_KEY_ID=your_access_key
ANTERAJA_SECRET_ACCESS_KEY=your_secret_key
ANTERAJA_SECRET_KEY_ONDEL=your_webhook_secret

# JT Cargo
JTCARGO_API_KEY=your_jtcargo_key
JTCARGO_USERNAME=your_jtcargo_username
JTCARGO_PASSWORD=your_jtcargo_password
```

Authentication Keys

```
# Partner Integration Authentication
EATSOK_API_KEY=your_eatsok_api_key
EATSOK_SECRET_KEY_ONDEL=your_eatsok_secret
LINKED_ACCOUNT_SECRET_KEY=your_linked_secret
INTEGRATION_PROC_KEY=your_integration_key
```

Marketplace Webhooks

```
# Blibli
BLIBLI_WEBHOOK_KEY=your_blibli_webhook_key

# OnMarket
ONMARKET_WEBHOOK_KEY=your_onmarket_webhook_key
```

Payment Gateway

```
# Xendit
XENDIT_API_KEY=your_xendit_api_key
XENDIT_SECRET_KEY=your_xendit_secret
```

Optional Environment Variables

```
# Email Configuration (Nodemailer)
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USER=your_email@gmail.com
EMAIL_PASSWORD=your_app_password
```

```
EMAIL_FROM=OnDelivery <noreply@ondelivery.com>

# File Upload Limits
MAX_FILE_SIZE=10485760  # 10MB in bytes

# Logging
LOG_LEVEL=info  # error, warn, info, debug
```

Environment Variable Loading

The application uses `dotenv` for environment variable management:

```
require('dotenv').config();
```

Best Practices:

- Use `.env` file for local development
- Use environment-specific `.env.production`, `.env.development`
- Never commit `.env` files to version control
- Use `dotenv-vault` for secure environment management (as mentioned in README)

Vault Management:

```
# Push to dotenv-vault
npx dotenv-vault@latest push
```

Development Guidelines

Development Guidelines

Code Structure Conventions

Controller Pattern

```
// controllers/<feature>/<feature>.controller.js

exports.functionName = async (req, res) => {
  try {
    // 1. Extract parameters
    const { param1, param2 } = req.body;

    // 2. Validate input (AJV schema)
    const validate = ajv.compile(schema);
    if (!validate(req.body)) {
      return res.status(400).json({
        message: "Validation error",
        errors: validate.errors
      });
    }

    // 3. Business logic
    const result = await performOperation(param1, param2);

    // 4. Return response
    return res.status(200).json({
      message: "Success",
      data: result
    });

  } catch (error) {
    console.error(error);
    return res.status(500).json({
      message: "Internal server error"
```

```
    });
  }
};
```

Route Pattern

```
// routes/<feature>.routes.js

const controller = require('../controllers/<feature>/<feature>.controller');
const { authJwt } = require('../middleware');

module.exports = function(app) {
  app.post(
    '/api/<feature>/<action>',
    [authJwt.verifyToken],
    controller.functionName
  );
};
```

Model Pattern

```
// models/<model>.model.js

module.exports = (sequelize, Sequelize) => {
  const Model = sequelize.define("model_name", {
    id: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
    field1: {
      type: Sequelize.STRING,
      allowNull: false
    },
    created_at: {
      type: Sequelize.DATE,
      defaultValue: Sequelize.NOW
    }
  }, {
    tableName: 'table_name',
```

```
    timestamps: false
  });

  return Model;
};
```

Naming Conventions

Files:

- Controllers: `<feature>.controller.js`
- Models: `<model>.model.js`
- Routes: `<feature>.routes.js`
- Utilities: `<purpose>.util.js`

Functions:

- camelCase for function names
- Descriptive names: `createWaybill`, `calculateFee`, `updateScan`

Variables:

- camelCase for variables
- Constants: UPPER_SNAKE_CASE

Database:

- Table names: snake_case, plural
- Column names: snake_case

Error Handling

Standard Error Response:

```
{
  "message": "Error description",
  "error": "Error code or type",
  "details": { /* additional context */ }
}
```

HTTP Status Codes:

- 200: Success

- 201: Created
- 400: Bad Request (validation errors)
- 401: Unauthorized (authentication failed)
- 403: Forbidden (insufficient permissions)
- 404: Not Found
- 500: Internal Server Error

Error Handler Middleware:

Location: `app/middleware/error_param_handler.js`

Usage: Automatically applied to all routes

Input Validation

AJV Schema Example:

```
// schemas/<feature>.schema.js

const createWaybillSchema = {
  type: "object",
  properties: {
    origin_code: { type: "string", minLength: 3 },
    destination_code: { type: "string", minLength: 3 },
    weight: { type: "number", minimum: 0 },
    service_type: { type: "string", enum: ["REG", "EXP", "CARGO"] }
  },
  required: ["origin_code", "destination_code", "weight", "service_type"],
  additionalProperties: false
};

module.exports = { createWaybillSchema };
```

Usage in Controller:

```
const ajv = new Ajv();
const { createWaybillSchema } = require('../schemas/waybill.schema');

const validate = ajv.compile(createWaybillSchema);
if (!validate(req.body)) {
  return res.status(400).json({
```

```
    message: "Validation failed",
    errors: validate.errors
  });
}
```

Database Best Practices

Transactions:

```
const t = await db.sequelize.transaction();

try {
  // Multiple operations
  await Waybill.create(data, { transaction: t });
  await Scan.create(scanData, { transaction: t });

  await t.commit();
} catch (error) {
  await t.rollback();
  throw error;
}
```

Query Optimization:

```
// Use specific fields instead of SELECT *
await Waybill.findAll({
  attributes: ['id', 'awb_number', 'status'],
  where: { status: 'BOOKED' },
  limit: 100
});

// Use includes for relationships
await Waybill.findOne({
  where: { id: waybillId },
  include: [
    { model: Scan, as: 'scans' },
    { model: Partner, as: 'partner' }
  ]
});
```

Logging Best Practices

Console Logging:

```
// Use cli-color for important logs
const clc = require('cli-color');

console.log(clc.green('Success: Waybill created'));
console.log(clc.yellow('Warning: Weight discrepancy detected'));
console.log(clc.red('Error: Database connection failed'));
```

Morgan HTTP Logging:

Already configured in `ondeliv-backend.js` with custom colored format.

Testing

Manual Testing Scripts:

- `check_fee_test.js` - Test fee calculation
- `check_price.js` - Test pricing lookup
- `debug_provinces.js` - Debug location data

Run Tests:

```
node check_fee_test.js
node check_price.js
```

Git Workflow

Branch Strategy (from README.md):

- `master/main` - Production branch
- `demo/trial` - Demo environment
- `development` - Development branch
- `hotfix` - Quick fixes from production
- `features` - New features
- `release` - Production-ready staging
- `experimental` - Experimental features

Commit Messages:

```
feat: Add new courier partner integration
fix: Resolve fee calculation bug
docs: Update API documentation
refactor: Improve waybill creation logic
test: Add unit tests for scanning operations
```

Code Review Checklist

Before submitting code:

- [ ] Code follows naming conventions
- [ ] Input validation implemented (AJV schemas)
- [ ] Error handling implemented
- [ ] Database transactions used where needed
- [ ] Authentication/authorization checked
- [ ] Logging added for important operations
- [ ] No sensitive data in logs
- [ ] No hardcoded credentials
- [ ] Comments added for complex logic
- [ ] Tested manually with sample data

Troubleshooting

Common Issues and Solutions

1. Database Connection Failed

Error:

```
Unable to connect Osas database: SequelizeConnectionError
```

Solutions:

- Check PostgreSQL is running: `systemctl status postgresql`
- Verify database credentials in `.env`
- Ensure database exists: `psql -U postgres -l`
- Check network connectivity to database server
- Verify firewall allows PostgreSQL port (5432)

2. JWT Token Invalid

Error:

```
{
  "message": "Unauthorized",
  "error": "Invalid token"
}
```

Solutions:

- Verify JWT_SECRET in `.env` matches token generation
- Check token expiration (JWT_EXPIRATION setting)
- Ensure Authorization header format: `Bearer <token>`
- Generate new token via `/api/auth/signin`

3. Rate Limit Exceeded

Error:

```
{
  "message": "You are sending requests too quickly. Please wait 1 second."
}
```

Solutions:

- Wait for rate limit window to reset
- Adjust RATE_LIMIT_WINDOW_MS and RATE_LIMIT_MAX in `.env`
- Implement request queuing in client
- Use batch endpoints where available

4. File Upload Failed

Error:

```
{
  "message": "File too large"
}
```

Solutions:

- Check file size < 10MB
- Verify `resources/temp/img/` directories exist
- Check disk space: `df -h`
- Ensure correct multipart/form-data encoding

5. Courier Webhook Not Working

Error: Webhooks not updating status

Solutions:

- Verify courier credentials in `.env`
- Check webhook endpoint is accessible (not localhost)
- Validate webhook authentication (Basic Auth, API Key)
- Check courier status mapping in `app/status/`
- Review webhook logs in console
- Test with webhook testing tools (ngrok, webhook.site)

6. Fee Calculation Returns 0

Error:

```
{
  "fee": 0
}
```

Solutions:

- Verify origin/destination codes are valid
- Check location data in database
- Ensure service availability for route
- Review pricing configuration
- Check for special pricing rules
- Validate weight parameter

7. Sequelize Migration Issues

Error:

```
SequelizeDatabaseError: relation does not exist
```

Solutions:

- Run migrations: Check for migration scripts
- Sync models: `db.sequelize.sync({ alter: true })`
- Check model definitions match database schema
- Verify table names in model files

8. CORS Errors

Error:

```
Access to fetch blocked by CORS policy
```

Solutions:

- Add origin to ORIGIN_CORS environment variable
- Verify CORS configuration in `ondeliv-backend.js`
- Check request includes proper headers
- Ensure preflight OPTIONS requests are handled

9. Image Not Loading

Error: 404 on image URLs

Solutions:

- Verify image directories exist:
  - `resources/temp/img/pickup/`
  - `resources/temp/img/delivery/`
  - `resources/temp/img/agent/`
- Check file permissions
- Verify image processing with Sharp
- Check BASEURL environment variable

10. Cron Job Not Running

Error: Lion Parcel pickup not scheduling

Solutions:

- Verify LIONPARCEL_PICKUP_CRON format is valid
- Check cron expression: `*/30 * * * *`
- Review cron job logs in console
- Ensure Lion Parcel credentials configured
- Test cron expression: https://crontab.guru

Debug Mode

Enable Verbose Logging:

```
// Add to ondeliv-backend.js temporarily
app.use((req, res, next) => {
  console.log(`${req.method} ${req.url}`);
  console.log('Headers:', req.headers);
  console.log('Body:', req.body);
  next();
});
```

Database Query Logging:

```
// In db.config.js
module.exports = {
  // ... existing config
  logging: console.log,  // Enable SQL query logging
};
```

Health Check Endpoint

Check Server Status:

```
curl http://localhost:4220/
```

Expected Response:

```
{
  "message": "This backend osas works well for now"
}
```

Performance Monitoring

Check Database Connection:

```
// Test database connectivity
const testConnection = async () => {
  try {
    await db.sequelize.authenticate();
    console.log('Database connected');
  } catch (error) {
    console.error('Database connection failed:', error);
  }
};
```

Monitor Response Times:

Morgan logs include response time in cyan color.

Getting Help

Resources:

- Review error logs in console
- Check courier partner documentation
- Review Sequelize documentation for database issues
- Check Express.js documentation for routing issues

Contact:

- Check repository issues
- Review pull request discussions

- Contact system administrator

Additional Resources

Key Files for Reference

- `ondeliv-backend.js` - Main application entry point
- `app/models/index.js` - Database model aggregation
- `app/middleware/authJwt.js` - Authentication logic
- `app/utility/check-fee.util.js` - Fee calculation logic
- `app/utility/waybill.util.js` - Waybill utilities

Third-Party Documentation

- Express.js - Web framework
- Sequelize - ORM documentation
- Passport.js - Authentication
- AJV - JSON Schema validation
- JWT - JSON Web Tokens

Recommended Reading

1. Start with `ondeliv-backend.js` to understand application initialization
2. Review `app/routes/auth.routes.js` for authentication flow
3. Study `app/controllers/waybill/create-waybill.controller.js` for core business logic
4. Examine `app/models/scans.model.js` for tracking data structure
5. Check `app/controllers/3p/` for courier integration patterns

Appendix

Status Code Reference

Code	Description	Usage
BKD	Booked	Initial waybill creation
PKD	Packed	Package prepared for shipping
HND	Handover	Transferred to courier/hub
DPT	Departure	Left origin hub
TRS	Transit	In transit between hubs
ARV	Arrival	Arrived at destination hub
RCV	Receiving	Received at DC
OFD	Out for Delivery	With delivery driver
DLV	Delivered	Successfully delivered
PRB	Problem	Issue encountered
RTN	Return	Package being returned
CNL	Cancel	Shipment cancelled

Service Type Codes

Code	Description
REG	Regular Service
EXP	Express Service
CARGO	Cargo Service
SAME	Same Day Delivery
NEXT	Next Day Delivery

Problem Type Codes

Code	Description
DMG	Damaged
LST	Lost
RFS	Refused by Recipient
INC	Incomplete Address
CLO	Closed/Not Available
WTH	Weather Issues
OTH	Other