Development Guidelines

- Development Guidelines
- Troubleshooting
- Additional Resources

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