Backend

- Introduction
  - Project Overview
  - Architecture
- Development guidelines
  - Getting started
  - Environment Configuration
  - Database Setup
  - Project structure
  - Core modules
  - API Documentation
  - Authentication & Authorization
  - Features & Integrations
  - Scheduled Tasks
  - WebSocket and payment integration
  - File Upload & Storage
  - Testing and deployment
  - Troubleshooting
  - Best practices
  - Additional Resources
  - Appendix

Introduction

Project Overview

OnMarket Landing API is a comprehensive e-commerce backend framework built with Node.js and Express.js. It provides a complete marketplace solution with multi-vendor support, affiliate marketing, payment processing, order management, and real-time chat functionality.

Key Capabilities

- **Multi-vendor Marketplace**: Support for multiple stores with individual management
- **Product Management**: Complete CRUD operations with variants, categories, and inventory
- **Order Processing**: Full order lifecycle from cart to delivery tracking
- **Payment Gateway**: Integrated with Xendit for multiple payment methods
- **Affiliate Marketing**: 3-level referral system with commission tracking
- **Shipping Integration**: GoSend instant delivery and other courier services
- **Real-time Chat**: WebSocket-based customer-seller communication
- **Virtual Accounts**: Automated payment collection via bank virtual accounts
- **Review System**: Product reviews with media uploads
- **CMS Features**: Content management for banners, categories, and promotions

---

Tech Stack

Core Technologies

- **Runtime**: Node.js
- **Framework**: Express.js v4.21.0
- **Language**: JavaScript (ES6+)

Databases

- **PostgreSQL**: Primary relational database (via Sequelize ORM v6.37.3)
- **MongoDB**: Document storage for chats, affiliate data, and caching (via Mongoose v8.6.3)
- **Redis**: Caching and session management (v4.7.0)

Key Dependencies

Web Framework & Middleware

- `express` - Web application framework
- `cors` - Cross-origin resource sharing
- `helmet` - Security headers
- `morgan` - HTTP request logger
- `express-validator` - Request validation
- `passport` - Authentication middleware

Data Management

- `sequelize` - PostgreSQL ORM
- `mongoose` - MongoDB ODM
- `redis` - Redis client
- `pg` - PostgreSQL driver

Authentication & Security

- `jsonwebtoken` - JWT token generation
- `jwt-decode` - JWT token decoding
- `passport-http` - HTTP authentication strategy
- `helmet` - Security middleware

File Processing

- `multer` - File upload handling
- `minio` - Object storage (S3-compatible)
- `sharp` - Image processing
- `ffmpeg-static` & `fluent-ffmpeg` - Video processing
- `bwip-js` - Barcode generation

Document Generation

- `pdfkit` - PDF generation
- `pdfmake` - Advanced PDF creation
- `exceljs` - Excel file generation

External Integrations

- `xendit-node` - Payment gateway integration
- `axios` - HTTP client
- `nodemailer` - Email sending

- `socket.io` - WebSocket for real-time features

Task Scheduling

- `node-cron` - Cron job scheduling
- `bull` - Job queue management

Utilities

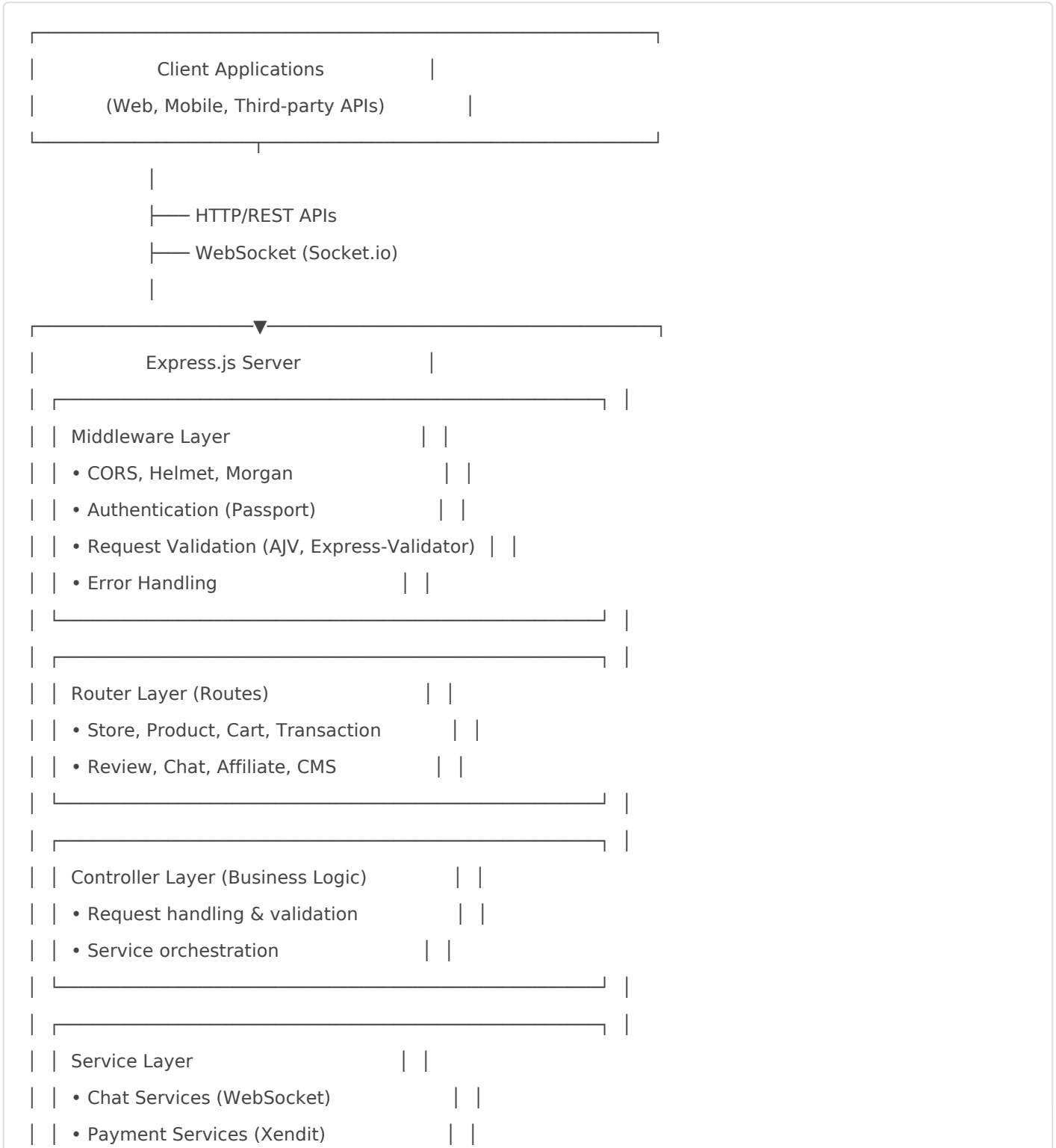
- `dayjs` - Date manipulation
- `lodash` - Utility functions
- `nanoid` - Unique ID generation
- `uuid` - UUID generation
- `ajv` - JSON schema validation
- `cli-color` - Console color output
- `dotenv` - Environment variable management

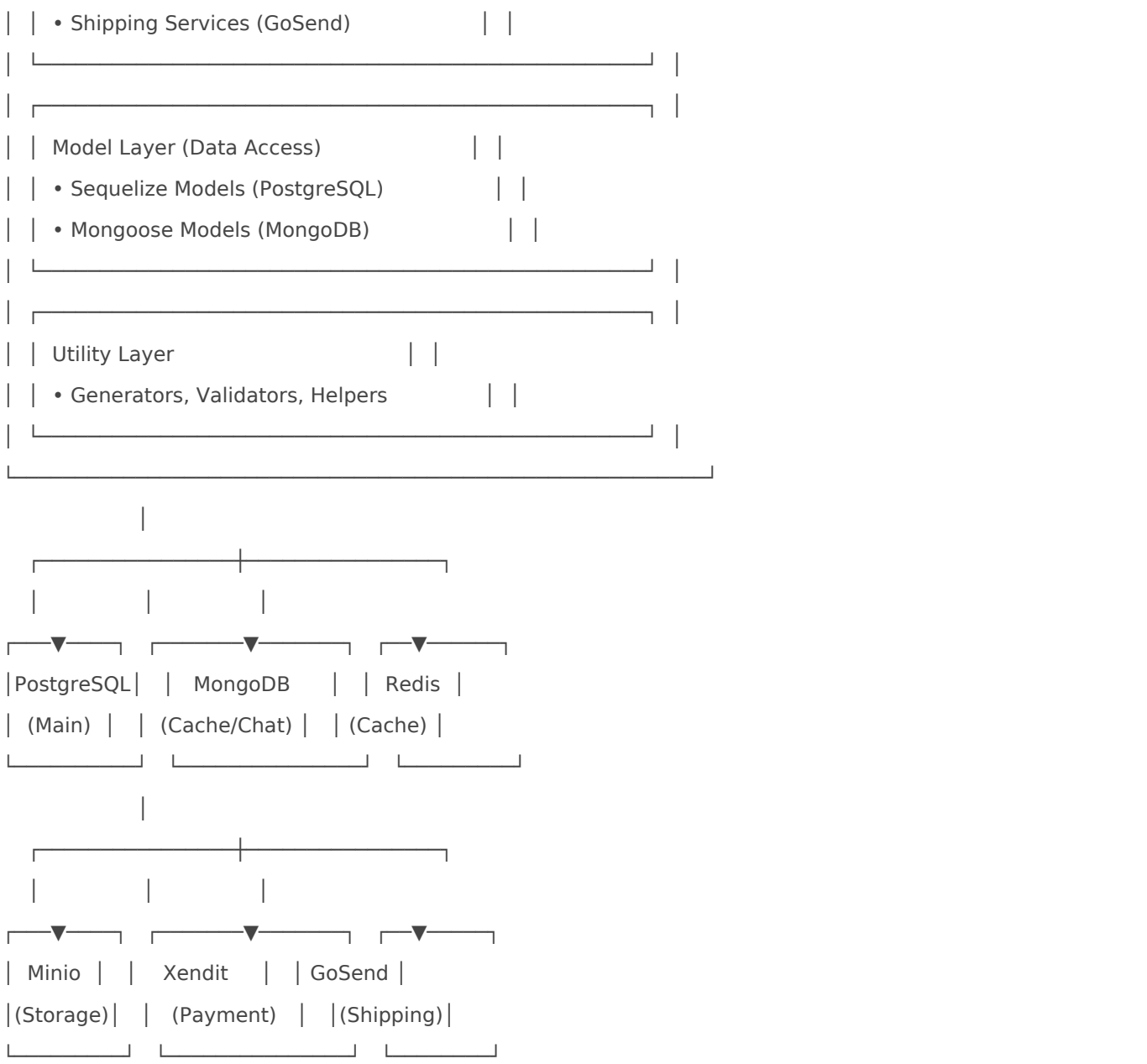
API Documentation

- `@scalar/express-api-reference` - Interactive API documentation

Architecture

Application Architecture





Request Flow

1. **Client Request** → Express server receives HTTP/WebSocket request
2. **Middleware** → Authentication, validation, logging
3. **Router** → Route matching and handler selection
4. **Controller** → Business logic execution
5. **Service/Model** → Data persistence and external API calls
6. **Response** → JSON response or WebSocket event emission

Development guidelines

Getting started

Prerequisites

- **Node.js**: v14.x or higher (LTS recommended)
- **npm**: v6.x or higher (comes with Node.js)
- **PostgreSQL**: v12.x or higher
- **MongoDB**: v4.x or higher
- **Redis**: v6.x or higher
- **Minio** (optional): For S3-compatible object storage

Installation Steps

1. **Clone the repository**

```
git clone https://github.com/ondeliveroper/on-market-landing-api.git
cd on-market-landing-api
```

2. **Install dependencies**

```
npm install
```

3. **Configure environment variables**

```
cp .env.example .env
# Edit .env with your configuration
```

4. **Set up databases**
   - Create PostgreSQL database
   - Start MongoDB service
   - Start Redis service
   - Run migrations (if any)
5. **Create required directories**

```
mkdir -p resources/temp/upload
mkdir -p resources/temp/review
mkdir -p resources/temp/pdf
```

6. **Start the development server**

```
npm start
```

7. **Access the API**
   - API Base: `http://localhost:3610`
   - API Documentation: `http://localhost:3610/api/kitab-suci` (development mode only)

Environment Configuration

Required Environment Variables

Create a `.env` file in the root directory with the following variables:

```
# Application
ENV=DEVELOPMENT  # or PRODUCTION
API_PORT=3610
BASE_DOMAIN=https://api.yourdomain.com
ORIGIN_CORS=https://yourdomain.com

# Directory Configuration
DIR=resources/upload

# Database - PostgreSQL
DB_HOST=localhost
DB_PORT=5432
DB_USER=your_db_user
DB_PASSWORD=your_db_password
DB=onmarket_db

# Database - MongoDB
MONGO_DB=mongodb://localhost:27017/onmarket

# Redis
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=your_redis_password

# Authentication
JWT_SECRET=your_jwt_secret_key
BASIC_AUTH_USER=admin
BASIC_AUTH_PASS=your_basic_auth_password
```

```
# Minio (Object Storage)
MINIO_ENDPOINT=localhost
MINIO_PORT=9000
MINIO_ACCESS_KEY=your_minio_access_key
MINIO_SECRET_KEY=your_minio_secret_key
MINIO_USE_SSL=false
MINIO_BUCKET=onmarket

# Payment Gateway - Xendit
XENDIT_SECRET_KEY=your_xendit_secret_key
XENDIT_CALLBACK_TOKEN=your_xendit_callback_token
XENDIT_PUBLIC_KEY=your_xendit_public_key

# Shipping - GoSend
GOSEND_CLIENT_ID=your_gosend_client_id
GOSEND_KEY=your_gosend_api_key
GOSEND_ADDRESS=https://integration-kilat-api.gojekapi.com/gokilat/v10

# Email Configuration
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USER=your_email@gmail.com
EMAIL_PASSWORD=your_email_password
EMAIL_FROM=noreply@onmarket.com

# Encryption
AES_SECRET_KEY=your_aes_encryption_key

# API Keys
WHATSAPP_API_KEY=your_whatsapp_api_key
WHATSAPP_API_URL=https://api.whatsapp.com

# Bull Queue (Redis-based)
BULL_REDIS_HOST=localhost
BULL_REDIS_PORT=6379
```

Environment-Specific Settings

Development

```
ENV=DEVELOPMENT
BASE_DOMAIN=http://localhost:3610
# Enables Scalar API documentation at /api/kitab-suci
```

Production

```
ENV=PRODUCTION
BASE_DOMAIN=https://api.yourdomain.com
# API documentation disabled
# Enhanced security headers
# Performance optimizations
```

Database Setup

PostgreSQL Setup

1. **Create Database**

```
CREATE DATABASE onmarket_db;
CREATE USER onmarket_user WITH ENCRYPTED PASSWORD 'your_password';
GRANT ALL PRIVILEGES ON DATABASE onmarket_db TO onmarket_user;
```

2. **Database Models** The application uses Sequelize ORM with auto-sync. Models include:
   - `store` - Store information
   - `user_info` - User profiles
   - `product` - Product catalog
   - `cart` - Shopping carts
   - `store_order` - Orders
   - `balance_mutation` - Financial transactions
   - `affiliate_account` - Affiliate accounts
   - `affiliate_commissions` - Commission records
   - `address_list` - Delivery addresses
   - `bank_info` - Bank account information
   - `review` - Product reviews
   - `voucher` - Discount vouchers
   - And many more...
3. **Initial Data**
   - Categories
   - Expedition methods
   - Bank configurations
   - Commission rates

MongoDB Setup

1. **Start MongoDB**

```
mongod --dbpath /path/to/data/directory
```

2. **Collections** MongoDB is used for:
   - `store_chat` - Chat messages
   - `store_chat_room` - Chat rooms

- `affiliate_category` - Affiliate categories
- `affiliate_clicks` - Click tracking
- `affiliate_links` - Affiliate link management
- `order_info_cached` - Order cache
- `shipping_info_cached` - Shipping cache
- `price_info_cached` - Price cache
- `expedition_count` - Delivery statistics

Redis Setup

1. **Start Redis**

   ```
   redis-server
   ```

2. **Usage** Redis is used for:
   - Session management
   - API response caching
   - Rate limiting
   - Bull queue jobs

Project structure

```
on-market-landing-api/
│
├── config/                 # Configuration files
│   ├── api.js              # API endpoint configurations
│   ├── bull.js             # Bull queue configuration
│   ├── db.js               # Database configuration
│   ├── minio.js            # Minio storage configuration
│   ├── multer.js           # File upload configuration
│   └── redis.js            # Redis client configuration
│
├── controller/             # Request handlers & business logic
│   ├── affiliate/          # Affiliate management controllers
│   ├── transaction/        # Transaction processing
│   │   ├── checkout.controller.js
│   │   ├── order_management.controller.js
│   │   ├── transaction_admin.controller.js
│   │   ├── transaction_report.controller.js
│   │   └── update_status.controller.js
│   ├── virtualAccount/     # Virtual account management
│   ├── auth_complement.controller.js
│   ├── balance.controller.js
│   ├── bank.controller.js
│   ├── cart.controller.js
│   ├── chat.controller.js
│   ├── cms.controller.js
│   ├── expedition.controller.js
│   ├── osas.controller.js
│   ├── product.controller.js
│   ├── review.controller.js
│   ├── store.controller.js
│   ├── trend.controller.js
│   ├── voucher.controller.js
│   ├── webhook.controller.js
│   └── wishlist.controller.js
```

```
│
├── docs/                    # API documentation
│   └── openapi.json          # OpenAPI 3.0 specification
│
├── middleware/              # Express middleware
│   ├── auth.js              # Authentication middleware
│   ├── encode_json_from_form-data.js
│   ├── error_param_handler.js   # Error handling
│   ├── json_validator.js        # JSON validation
│   ├── multer_additional.js     # File upload helpers
│   ├── other_validator.js       # Custom validators
│   ├── passport.js              # Passport strategies
│   ├── review.middleware.js     # Review-specific middleware
│   ├── safecheck.js             # Security checks
│   ├── signature.js             # Request signature validation
│   ├── utility.js               # Utility middleware
│   └── verify.js                # Token verification
│
├── model/                   # Data models
│   ├── affiliate/           # Affiliate-related models
│   ├── cms/                 # CMS models
│   ├── config_expedition/       # Shipping configuration
│   ├── master_location/         # Location data
│   ├── mongodb/                 # MongoDB schemas
│   ├── pricelist/           # Pricing models
│   ├── index.js             # Model initialization
│   └── [various model files]    # Sequelize models
│
├── resources/               # Static resources & uploads
│   ├── static/              # Static files
│   └── temp/                # Temporary files
│       ├── upload/          # Upload temp directory
│       ├── review/          # Review image temp
│       └── pdf/             # PDF generation temp
│
├── router/                  # Route definitions
│   ├── address.routes.js
│   ├── affiliate.routes.js
│   ├── auth_complement.routes.js
│   ├── balance.routes.js
```

```
│   ├── bank.routes.js
│   ├── cart.routes.js
│   ├── chat.routes.js
│   ├── cms.routes.js
│   ├── expedition.routes.js
│   ├── osas.routes.js
│   ├── product.routes.js
│   ├── review.routes.js
│   ├── store.routes.js
│   ├── transaction.routes.js
│   ├── transaction_report.routes.js
│   ├── trend.routes.js
│   ├── virtual_account.routes.js
│   ├── voucher.routes.js
│   └── wishlist.routes.js
│
├── schedulers/                # Scheduled tasks
│   └── scheduledTask.js       # Cron job definitions
│
├── schema/                    # Validation schemas
│   ├── address.schema.js
│   ├── affiliate.schema.js
│   ├── balance.schema.js
│   ├── bank.schema.js
│   ├── cart.schema.js
│   ├── expedition.schema.js
│   ├── index.js
│   ├── osas.schema.js
│   ├── product.schema.js
│   ├── review.schema.js
│   ├── review-order.schema.js
│   ├── schedule.schema.js
│   ├── transaction.schema.js
│   ├── transaction-report.schema.js
│   ├── update_expedition_service.schema.js
│   ├── voucher.schema.js
│   ├── webhook.schema.js
│   └── wishlist.schema.js
│
├── scripts/                   # Utility scripts
```

```
│   └── daily_liquidation.js     # Daily settlement script
│
├── services/                    # External service integrations
│   ├── briva/                   # BRI Virtual Account
│   └── chat.services.js         # Chat WebSocket handling
│
├── utility/                     # Helper functions
│   ├── determineShippingService.utility.js
│   ├── email_util.js
│   ├── general.utility.js
│   ├── generateDailyVouchers.js
│   ├── generateHmacSignature.js
│   ├── generateSignatureToken.js
│   ├── generator.utility.js
│   ├── payment_util.js
│   ├── pick_expedition.utility.js
│   ├── product_validator.js
│   ├── psql.utility.js
│   ├── shipment.utility.js
│   ├── validate_signature.js
│   └── whatsappapi.js
│
├── .gitignore                   # Git ignore rules
├── DOCUMENTATION.md             # This file
├── GOSEND_INTEGRATION_README.md # GoSend integration guide
├── index.js                     # Application entry point
├── package.json                 # Dependencies & scripts
├── package-lock.json            # Locked dependencies
├── README.md                    # Project readme
├── REFERRAL_SYSTEM_README.md    # Referral system guide
├── referral_system_test.js      # Referral system tests
└── utility.js                   # Legacy utility file
```

Core modules

1. Store Management

- Store registration and activation
- Store profile management
- Store schedule configuration
- Online/offline status
- Store front customization

Key Files:

- `controller/store.controller.js`
- `router/store.routes.js`
- `model/store.model.js`

2. Product Management

- Product CRUD operations
- Product variants
- Inventory management
- Category management
- Product photos (up to 20 images)
- POS integration

Key Files:

- `controller/product.controller.js`
- `router/product.routes.js`
- `model/product.model.js`

3. Shopping Cart

- Add/remove items
- Update quantities
- Cart listing
- Invalid item detection

Key Files:

- `controller/cart.controller.js`
- `router/cart.routes.js`
- `model/cart.model.js`

4. Order Processing

- Checkout flow
- Order creation
- Order status updates
- Order management (seller side)
- Order tracking
- Delivery monitoring
- Invoice generation

Key Files:

- `controller/transaction/checkout.controller.js`
- `controller/transaction/order_management.controller.js`
- `controller/transaction/update_status.controller.js`

5. Payment System

- Xendit integration
- Virtual account generation
- Payment callbacks
- Payment verification
- Balance management
- Payout processing

Key Files:

- `controller/virtualAccount/virtual_account.controller.js`
- `controller/balance.controller.js`
- `controller/webhook.controller.js`

6. Affiliate Marketing

- 3-level referral system
- Affiliate link generation
- Click tracking
- Commission calculation

- Referral bonus distribution (20% per level)
- Affiliate dashboard

Key Files:

- `controller/affiliate/`
- `router/affiliate.routes.js`
- See `REFERRAL_SYSTEM_README.md` for details

7. Shipping & Expedition

- Multiple courier support
- GoSend Instant integration
- Shipping cost calculation
- Real-time price API
- Order booking
- Delivery tracking

Key Files:

- `controller/expedition.controller.js`
- `utility/shipment.utility.js`
- See `GOSEND_INTEGRATION_README.md` for details

8. Review System

- Product reviews
- Rating system
- Photo/video uploads
- Review moderation
- Order review

Key Files:

- `controller/review.controller.js`
- `router/review.routes.js`

9. Real-time Chat

- Customer-seller messaging
- WebSocket communication
- Chat rooms
- Message history

- Unread message tracking

Key Files:

- `services/chat.services.js`
- `controller/chat.controller.js`
- `model/mongodb/store_chat.model.js`

10. CMS

- Banner management
- Category management
- Content pages
- Promotional content

Key Files:

- `controller/cms.controller.js`
- `model/cms/`

API Documentation

Interactive API Documentation

When running in `DEVELOPMENT` mode, access interactive API documentation at:

```
http://localhost:3610/api/kitab-suci
```

This uses Scalar API Reference with the OpenAPI 3.0 specification defined in `docs/openapi.json`.

Authentication Methods

The API supports two authentication methods:

1. Bearer Token (JWT)

```
Authorization: Bearer <jwt_token>
```

2. Basic Auth

```
Authorization: Basic <base64(username:password)>
```

Common API Endpoints

Store Management

```
GET  /verify/store/exist          # Check if store exists
GET  /store/activate              # Activate store
POST /store/edit                  # Update store info
POST /store/activity/edit         # Update store activity hours
POST /store/schedule/edit         # Update store schedule
GET  /store/check-online-status   # Check online status
```

Product Management

```
POST /product/add                # Add new product
POST /product/edit               # Edit product
POST /product/delete             # Delete product
POST /product/list               # List products
GET  /product/detail/:id         # Get product details
```

Cart Operations

```
GET  /cart/list                  # Get cart items
POST /cart/add                   # Add item to cart
POST /cart/update                # Update cart item
POST /cart/delete                # Remove from cart
```

Checkout & Orders

```
POST /transaction/review-order      # Review order before checkout
POST /transaction/checkout          # Create order
GET  /transaction/order/list        # List orders
GET  /transaction/order/:id         # Get order details
POST /transaction/order/confirm-shipment  # Confirm shipment
GET  /transaction/order/monitor-delivery  # Track delivery
```

Payments

```
POST /virtual-account/create        # Create virtual account
GET  /virtual-account/list          # List VAs
POST /webhook/xendit                # Xendit payment webhook
GET  /balance/mutation              # Balance history
```

Affiliate

```
POST /affiliate/register            # Register as affiliate
GET  /affiliate/links               # Get affiliate links
GET  /affiliate/dashboard/stats     # Affiliate statistics
GET  /affiliate/dashboard/referral-stats  # Referral stats
GET  /affiliate/commissions         # Commission history
```

Reviews

```
POST /review/add                # Add product review
GET  /review/list               # List reviews
POST /review/reply              # Reply to review
```

Chat

```
GET  /chat/rooms                # List chat rooms
GET  /chat/messages/:roomId     # Get messages
POST /chat/send                 # Send message (via WebSocket)
```

Response Format

Success Response

```
{
  "success": true,
  "message": "Operation completed successfully",
  "data": {
    // Response data
  }
}
```

Error Response

```
{
  "success": false,
  "message": "Error message",
  "errors": [
    {
      "field": "fieldName",
      "message": "Validation error message"
    }
  ]
}
```

Authentication & Authorization

JWT Authentication

1. **Login Flow**
   - User authenticates through external auth service
   - Receives JWT token
   - Token contains: `user_id`, `username`, `store_id` (if applicable)
2. **Token Usage**

```
// Token verification in middleware
const token = req.headers.authorization?.split(' ')[1];
const decoded = jwt.verify(token, process.env.JWT_SECRET);
```

3. **Token Refresh**
   - Implement token refresh logic in auth service
   - Tokens typically expire after 24 hours

Passport Strategies

Basic HTTP authentication for admin/internal endpoints:

```
// In middleware/passport.js
passport.use(new BasicStrategy(
  function(username, password, done) {
    // Validate credentials
  }
));
```

Authorization Levels

1. **Public** - No authentication required
2. **Authenticated User** - Requires valid JWT

3. **Store Owner** - Requires JWT with store_id
4. **Admin** - Requires Basic Auth or special admin JWT
5. **Affiliate** - Requires JWT with affiliate permissions

Middleware Chain Example

```
app.post(
  '/product/add',
  verifyCross,           // Verify JWT token
  getStoreID,            // Extract store ID
  validator.validate(),  // Validate request
  productController.addProduct
);
```

Features & Integrations

1. Three-Level Referral System

Purpose: Incentivize user acquisition through multi-level referral bonuses.

Structure:

- Level 1: 20% bonus from product price
- Level 2: 20% bonus from product price
- Level 3: 20% bonus from product price

Implementation:

- See `REFERRAL_SYSTEM_README.md` for complete details
- Automatic bonus distribution on purchase
- Tax calculation (2.5% on bonuses)

2. Payment Gateway - Xendit

Supported Payment Methods:

- Virtual Accounts (BCA, BNI, BRI, Mandiri, Permata)
- E-wallets (OVO, Dana, LinkAja)
- Credit/Debit Cards
- Retail Outlets (Alfamart, Indomaret)

Flow:

1. Create payment request via Xendit API
2. Return payment instructions to user
3. Receive webhook on payment success
4. Update order status automatically

Files:

- `controller/virtualAccount/virtual_account.controller.js`
- `controller/webhook.controller.js`

3. File Upload & Media Processing

Supported Operations:

- Image upload (products, reviews, banners)
- Image compression with Sharp
- Video processing with FFmpeg
- PDF generation
- Barcode generation
- Minio object storage

Configuration:

- Max 20 images per product
- Image compression for web optimization
- Temporary file cleanup

4. Email Notifications

Triggers:

- Order confirmation
- Payment received
- Shipment confirmed
- Order delivered
- Review reminders

Configuration:

```
// In utility/email_util.js
const transporter = nodemailer.createTransport({
  host: process.env.EMAIL_HOST,
  port: process.env.EMAIL_PORT,
  auth: {
    user: process.env.EMAIL_USER,
    pass: process.env.EMAIL_PASSWORD
  }
});
```

5. WhatsApp API Integration

Features:

- Order notifications
- Payment reminders
- Promotional messages

Files:

- `utility/whatsappapi.js`

Scheduled Tasks

The application runs several scheduled tasks defined in `schedulers/scheduledTask.js`:

1. Update Invoice Status

Schedule: Every 5 minutes **Purpose**: Check and update expired payment invoices

```
scheduledTask.updateInvoiceStatus();
```

2. Clear Temp Folder

Schedule: Daily at 2 AM **Purpose**: Clean up temporary files (uploads, reviews, PDFs)

```
scheduledTask.clearTempFolder();
```

3. Clear Idle Transactions

Schedule: Every 30 minutes **Purpose**: Clean up abandoned PostgreSQL transactions

```
scheduledTask.clearIdleInTransaction();
```

4. Auto Commission for Affiliates

Schedule: Daily at 3 AM **Purpose**: Process pending affiliate commissions

```
scheduledTask.autoCommissionAffiliate();
```

5. Daily Liquidation

Schedule: Daily at 1 AM **Purpose**: Settle vendor balances and payouts

```
scheduledTask.dailyLiquidation();
```

WebSocket and payment integration

WebSocket/Real-time Features

Socket.IO Implementation

Server Setup (in `index.js`):

```
const server = http.createServer(app);
const io = socketIo(server, {
  cors: { origin: "*" }
});
handleChat(io);
```

Chat System

Features:

- Real-time messaging between buyers and sellers
- Chat room management
- Online status
- Typing indicators
- Unread message count

Events:

- `connection` - Client connected
- `join_room` - Join specific chat room
- `send_message` - Send message
- `typing` - User typing indicator
- `read_messages` - Mark messages as read
- `disconnect` - Client disconnected

Implementation (`services/chat.services.js`):

```
io.on('connection', (socket) => {
  socket.on('join_room', (roomId) => {
    socket.join(roomId);
  });

  socket.on('send_message', async (data) => {
    // Save message to MongoDB
    // Emit to room participants
    io.to(roomId).emit('new_message', message);
  });
});
```

Payment Integration

Xendit Integration

1. Create Virtual Account

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

// Create VA
const response = await xenditClient.VirtualAccount.create({
  external_id: orderNumber,
  bank_code: 'BCA',
  name: customerName,
  expected_amount: totalAmount,
  expiration_date: expirationDate
});
```

2. Webhook Handling

```
// POST /webhook/xendit
app.post('/webhook/xendit', async (req, res) => {
  // Verify webhook signature
```

```
  const signature = req.headers['x-callback-token'];
  if (signature !== process.env.XENDIT_CALLBACK_TOKEN) {
    return res.status(401).json({ error: 'Unauthorized' });
  }

  // Process payment
  const { external_id, status } = req.body;
  if (status === 'PAID') {
    await updateOrderStatus(external_id, 'PAID');
  }

  res.json({ success: true });
});
```

BRI Virtual Account (BRIVA)

Additional integration for BRI-specific virtual accounts in `services/briva/`.

File Upload & Storage

Multer Configuration

Local Storage (temporary):

```
// config/multer.js
const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, req.tempFolder);  // Dynamic temp folder
  },
  filename: (req, file, cb) => {
    cb(null, `${Date.now()}-${file.originalname}`);
  }
});
```

Minio Object Storage

Configuration (`config/minio.js`):

```
const minioClient = new Minio.Client({
  endPoint: process.env.MINIO_ENDPOINT,
  port: parseInt(process.env.MINIO_PORT),
  useSSL: process.env.MINIO_USE_SSL === 'true',
  accessKey: process.env.MINIO_ACCESS_KEY,
  secretKey: process.env.MINIO_SECRET_KEY
});
```

Upload Flow:

1. File uploaded to temp directory via Multer
2. Image processed/compressed with Sharp
3. File uploaded to Minio
4. Temp file deleted
5. Minio URL stored in database

Image Processing

```
const sharp = require('sharp');

// Compress and resize
await sharp(inputPath)
  .resize(800, 800, { fit: 'inside' })
  .jpeg({ quality: 80 })
  .toFile(outputPath);
```

Testing and deployment

Testing

Current Test Setup

The project currently has limited automated tests. Main test file:

- `referral_system_test.js` - Tests for referral system calculations

Running Tests

```
npm test
```

Manual Testing

1. **Use Postman/Insomnia**
   - Import OpenAPI spec from `docs/openapi.json`
   - Test endpoints manually
2. **Use Scalar API Reference**
   - Access `http://localhost:3610/api/kitab-suci` in development
   - Test directly from browser
3. **WebSocket Testing**
   - Use Socket.IO client library
   - Or use tools like Socket.IO Client Tool

Recommended Test Coverage

Future test implementation should cover:

- Unit tests for utilities and services
- Integration tests for API endpoints
- E2E tests for critical flows (checkout, payment)

- Load testing for concurrent operations

Deployment

Production Checklist

1. **Environment Configuration**
   - ☐ Set `ENV=PRODUCTION`
   - ☐ Configure production database credentials
   - ☐ Set secure JWT secret
   - ☐ Configure production API keys (Xendit, GoSend)
   - ☐ Set CORS origins
   - ☐ Configure email settings
2. **Database**
   - ☐ Run database migrations
   - ☐ Set up database backups
   - ☐ Configure connection pooling
   - ☐ Set up read replicas (if needed)
3. **Security**
   - ☐ Enable HTTPS
   - ☐ Configure Helmet security headers
   - ☐ Set up rate limiting
   - ☐ Configure firewall rules
   - ☐ Set up SSL certificates
4. **Performance**
   - ☐ Enable Redis caching
   - ☐ Configure CDN for static assets
   - ☐ Set up load balancer
   - ☐ Configure process manager (PM2)
5. **Monitoring**
   - ☐ Set up logging (Winston/Bunyan)
   - ☐ Configure error tracking (Sentry)
   - ☐ Set up uptime monitoring
   - ☐ Configure performance monitoring (New Relic/DataDog)

Deployment Options

Option 1: VPS/Dedicated Server

```
# Install PM2
npm install -g pm2

# Start application
pm2 start index.js --name onmarket-api

# Configure PM2 startup
pm2 startup
pm2 save

# Monitor
pm2 monit
```

Option 2: Docker

Create `Dockerfile`:

```
FROM node:16-alpine

WORKDIR /app

COPY package*.json ./
RUN npm ci --only=production

COPY . .

EXPOSE 3610

CMD ["node", "index.js"]
```

Create `docker-compose.yml`:

```
version: '3.8'
services:
  api:
    build: .
```

```
    ports:
      - "3610:3610"
    env_file:
      - .env
    depends_on:
      - postgres
      - mongo
      - redis

  postgres:
    image: postgres:14
    environment:
      POSTGRES_DB: onmarket_db
      POSTGRES_USER: onmarket
      POSTGRES_PASSWORD: password
    volumes:
      - postgres-data:/var/lib/postgresql/data

  mongo:
    image: mongo:5
    volumes:
      - mongo-data:/data/db

  redis:
    image: redis:7-alpine
    command: redis-server --requirepass yourpassword

volumes:
  postgres-data:
  mongo-data:
```

Run with Docker:

```
docker-compose up -d
```

Option 3: Cloud Platforms

AWS:

- Use Elastic Beanstalk
- Or ECS with Fargate

- RDS for PostgreSQL
- DocumentDB for MongoDB
- ElastiCache for Redis

Google Cloud:

- Cloud Run
- Cloud SQL
- Memorystore

Heroku:

```
heroku create onmarket-api
heroku addons:create heroku-postgresql
heroku addons:create mongolab
heroku addons:create heroku-redis
git push heroku main
```

Nginx Configuration

```
server {
    listen 80;
    server_name api.yourdomain.com;

    location / {
        proxy_pass http://localhost:3610;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_cache_bypass $http_upgrade;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    # WebSocket support
    location /socket.io/ {
        proxy_pass http://localhost:3610;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
```

```
        proxy_set_header Host $host;
    }
}
```

Troubleshooting

Common Issues

1. Database Connection Errors

Problem: Cannot connect to PostgreSQL/MongoDB

```
ERROR NOT CONNECTED to the database!
```

Solutions:

- Verify database is running
- Check connection credentials in `.env`
- Ensure firewall allows database port
- Check database host/port configuration

2. Redis Connection Failed

Problem: Redis client connection error

```
Redis Error - Connection refused
```

Solutions:

- Start Redis server: `redis-server`
- Check Redis password configuration
- Verify Redis port (default: 6379)
- Check if Redis is running: `redis-cli ping`

3. Minio Connection Issues

Problem: Cannot connect to Minio

```
Minio Error - Connection refused
```

Solutions:

- Start Minio server

- Verify endpoint, port, and credentials
- Check bucket exists
- Ensure SSL configuration matches

4. File Upload Failures

Problem: Files not uploading or temp directory errors

Solutions:

- Ensure temp directories exist and have write permissions

```
mkdir -p resources/temp/upload
mkdir -p resources/temp/review
mkdir -p resources/temp/pdf
chmod -R 755 resources/
```

- Check disk space
- Verify Multer configuration

5. JWT Token Invalid

Problem: Authentication fails with valid-looking token

Solutions:

- Verify JWT_SECRET matches token generation
- Check token expiration
- Ensure token format: `Bearer <token>`
- Verify token signature

6. Scheduled Tasks Not Running

Problem: Cron jobs not executing

Solutions:

- Check node-cron syntax
- Verify server timezone
- Check logs for errors
- Ensure process stays running (use PM2)

7. WebSocket Connection Failed

Problem: Chat not working, Socket.IO errors

Solutions:

- Check CORS configuration
- Verify WebSocket port is open
- Check Nginx/proxy configuration for WebSocket support
- Ensure HTTP server is created correctly

8. Payment Webhook Not Receiving

Problem: Xendit webhooks not being received

Solutions:

- Verify webhook URL is publicly accessible
- Check callback token matches
- Examine webhook logs in Xendit dashboard
- Test with webhook testing tools
- Ensure HTTPS in production

Debugging Tips

1. **Enable Debug Logging**

```
// In index.js
const morgan = require('morgan');
app.use(morgan('combined'));
```

2. **Check Database Queries**

```
// Sequelize logging enabled by default in development
// Check console for SQL queries
```

3. **Monitor Process**

```
pm2 logs onmarket-api
pm2 monit
```

4. **Check Port Availability**

```
lsof -i :3610
netstat -tulpn | grep 3610
```

5. **Test Database Connections**

```
# PostgreSQL
psql -h localhost -U onmarket_user -d onmarket_db

# MongoDB
mongo mongodb://localhost:27017/onmarket

# Redis
redis-cli -h localhost -p 6379 ping
```

Development guidelines

Best practices

Code Organization

1. **Follow MVC Pattern**
   - Models: Data structures and database operations
   - Controllers: Business logic and request handling
   - Routes: Endpoint definitions
2. **Use Middleware Chains**
   - Authentication → Validation → Business Logic
   - Keep middleware focused and reusable
3. **Error Handling**
   - Always use try-catch blocks
   - Return consistent error formats
   - Log errors with context
4. **Validation**
   - Validate all inputs with JSON schemas
   - Use AJV for schema validation
   - Sanitize user inputs

Security

1. **Environment Variables**
   - Never commit `.env` files
   - Use strong secrets and passwords
   - Rotate credentials regularly
2. **Authentication**
   - Always verify JWT tokens
   - Implement rate limiting
   - Use HTTPS in production
3. **SQL Injection Prevention**
   - Use Sequelize parameterized queries
   - Never concatenate SQL strings
   - Validate and sanitize inputs
4. **XSS Prevention**
   - Sanitize HTML inputs
   - Use Content Security Policy headers
   - Escape output data

Performance

1. **Database Optimization**
   - Add indexes on frequently queried columns
   - Use connection pooling
   - Implement caching with Redis
2. **API Response Time**
   - Cache expensive operations
   - Use pagination for lists
   - Optimize database queries (avoid N+1)
3. **File Handling**
   - Compress images before storage
   - Clean up temp files regularly
   - Use CDN for static assets
4. **Monitoring**
   - Log slow queries
   - Monitor memory usage
   - Track API response times

Development Workflow

1. **Version Control**
   - Use feature branches
   - Write descriptive commit messages
   - Review code before merging
2. **Testing**
   - Write tests for new features
   - Test edge cases
   - Perform integration testing
3. **Documentation**
   - Keep API docs updated
   - Document complex business logic
   - Write clear code comments
4. **Code Quality**
   - Use consistent naming conventions
   - Follow JavaScript best practices
   - Use ESLint for code linting

Additional Resources

Related Documentation

- GoSend Integration Guide
- Referral System Guide
- OpenAPI Specification

External Documentation

Core Technologies

- Node.js Documentation
- Express.js Guide
- Sequelize ORM
- Mongoose ODM

Databases

- PostgreSQL Documentation
- MongoDB Manual
- Redis Documentation

Integrations

- Xendit API Documentation
- GoSend API Documentation
- Socket.IO Documentation

Tools & Libraries

- Multer Documentation
- Sharp Image Processing

- Bull Job Queue
- AJV JSON Schema Validator
- Passport.js

Support & Community

- **GitHub Repository**: ondeliveroper/on-market-landing-api
- **Issues**: Report bugs and feature requests on GitHub Issues
- **Email**: Contact development team for support

Changelog

Document major changes and version releases:

Version 1.0.0 (Current)

- Initial release
- Multi-vendor marketplace
- Payment gateway integration
- GoSend delivery integration
- 3-level referral system
- Real-time chat
- Affiliate marketing

Appendix

Environment Variable Reference

Variable	Type	Required	Description
ENV	string	Yes	Environment (DEVELOPMENT/PRODUCTION)
API_PORT	number	Yes	Server port (default: 3610)
BASE_DOMAIN	string	Yes	API base URL
DB_HOST	string	Yes	PostgreSQL host
DB_PORT	number	Yes	PostgreSQL port
DB_USER	string	Yes	PostgreSQL username
DB_PASSWORD	string	Yes	PostgreSQL password
DB	string	Yes	PostgreSQL database name
MONGO_DB	string	Yes	MongoDB connection string
REDIS_HOST	string	Yes	Redis host
REDIS_PORT	number	Yes	Redis port
REDIS_PASSWORD	string	No	Redis password
JWT_SECRET	string	Yes	JWT signing secret
XENDIT_SECRET_KEY	string	Yes	Xendit API secret key
XENDIT_CALLBACK_TOKEN	string	Yes	Xendit webhook token
GOSEND_CLIENT_ID	string	Yes	GoSend API client ID
GOSEND_KEY	string	Yes	GoSend API key
GOSEND_ADDRESS	string	Yes	GoSend API base URL
MINIO_ENDPOINT	string	Yes	Minio server endpoint
MINIO_PORT	number	Yes	Minio server port
MINIO_ACCESS_KEY	string	Yes	Minio access key
MINIO_SECRET_KEY	string	Yes	Minio secret key
EMAIL_HOST	string	Yes	SMTP host

Variable	Type	Required	Description
EMAIL_PORT	number	Yes	SMTP port
EMAIL_USER	string	Yes	SMTP username
EMAIL_PASSWORD	string	Yes	SMTP password

Database Schema Overview

Key PostgreSQL Tables

- **store**: Store information and settings
- **user_info**: User profiles and authentication
- **product**: Product catalog
- **product_variant**: Product variations (size, color, etc.)
- **store_order**: Order records
- **store_order_details**: Line items in orders
- **cart**: Shopping cart items
- **address_list**: Delivery addresses
- **balance_mutation**: Financial transactions
- **affiliate_account**: Affiliate user accounts
- **affiliate_commissions**: Commission records
- **affiliate_invitations**: Referral relationships
- **review**: Product reviews
- **voucher**: Discount codes
- **virtual_account**: Payment virtual accounts
- **xendit_invoice**: Payment invoices
- **bank_info**: Bank account information
- **payout_record**: Payout history

Key MongoDB Collections

- **store_chat**: Chat messages
- **store_chat_room**: Chat room metadata
- **affiliate_category**: Affiliate product categories
- **affiliate_clicks**: Click tracking for affiliate links
- **affiliate_links**: Generated affiliate links
- **order_info_cached**: Cached order data
- **shipping_info_cached**: Cached shipping calculations
- **price_info_cached**: Cached price calculations

API Response Status Codes

Code	Meaning	Usage

200	OK	Successful request
201	Created	Resource created successfully
400	Bad Request	Invalid request parameters
401	Unauthorized	Authentication required
403	Forbidden	Insufficient permissions
404	Not Found	Resource not found
422	Unprocessable Entity	Validation errors
500	Internal Server Error	Server error
503	Service Unavailable	Service temporarily unavailable