Backend

github: on-internal-api

- API Documentation
  - API Documentation (will be separated)
- Architecture
  - New Page (will be separated)
- Deployment
  - New Page

API Documentation

Complete API reference for the SUNSHINE (ON Internal API) system.

API Documentation (will be separated)

Table of Contents

- Overview
- Base URL
- Authentication
- Common Headers
- Response Formats
- Error Codes
- API Endpoints
  - Authentication
  - User Management
  - Employee Management
  - Asset Management
  - Attendance
  - Leave Management
  - Permit Management
  - Fleet Management
  - Ticketing
  - Meeting Room
  - Branding Approval
  - CMS
  - File Upload
  - Sales
  - Freelance
  - Onapps

Overview

The ON Internal API provides RESTful endpoints and WebSocket connections for managing various business operations. All endpoints return JSON responses and use standard HTTP methods.

Base URL

```
Development: http://localhost:3601
Production: https://api.yourdomain.com
```

All endpoints are prefixed with `/api` unless otherwise specified.

Authentication

Most endpoints require authentication using JWT (JSON Web Token).

Obtaining a Token

```
POST /api/auth/login
Content-Type: application/json

{
  "username": "user@example.com",
  "password": "your_password"
}
```

Response:

```
{
  "success": true,
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": 1,
    "username": "user@example.com",
    "name": "John Doe",
```

```
    "role": "admin"
  }
}
```

Using the Token

Include the token in the `x-access-token` header for protected endpoints:

```
GET /api/auth/account
x-access-token: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

Common Headers

Request Headers

```
Content-Type: application/json
x-access-token: <JWT_TOKEN>  # For authenticated endpoints
```

Response Headers

```
Content-Type: application/json
Cross-Origin-Resource-Policy: cross-origin
```

Response Formats

Success Response

```
{
  "success": true,
  "data": {
    // Response data
  },
```

```
  "message": "Operation successful"
}
```

Error Response

```
{
  "success": false,
  "message": "Error description",
  "errors": [
    {
      "field": "fieldName",
      "message": "Field-specific error"
    }
  ]
}
```

Paginated Response

```
{
  "success": true,
  "data": [...],
  "pagination": {
    "page": 1,
    "perPage": 20,
    "total": 150,
    "totalPages": 8
  }
}
```

Error Codes

Status Code	Description
200	Success
201	Created

Status Code	Description
400	Bad Request (validation error)
401	Unauthorized (authentication required)
403	Forbidden (insufficient permissions)
404	Not Found
500	Internal Server Error

API Endpoints

Authentication Endpoints

POST /api/auth/login

Authenticate a user and receive a JWT token.

Request Body:

```
{
  "username": "user@example.com",
  "password": "password123"
}
```

Response:

```
{
  "success": true,
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "user": {
    "id": 1,
    "username": "user@example.com",
    "name": "John Doe",
    "role": "admin"
  }
}
```

Status Codes:

- `200`: Login successful
- `401`: Invalid credentials

GET /api/auth/logout

Logout the current user.

Headers:

```
x-access-token: <JWT_TOKEN>
```

Response:

```
{
  "success": true,
  "message": "Logged out successfully"
}
```

GET /api/auth/account

Get the current authenticated user's account information.

Headers:

```
x-access-token: <JWT_TOKEN>
```

Response:

```
{
  "success": true,
  "data": {
    "id": 1,
    "username": "user@example.com",
    "name": "John Doe",
    "email": "user@example.com",
    "role": "admin",
    "permissions": [...]
```

```
  }
}
```

GET /api/auth/verify-jwt

Verify if the JWT token is valid.

Headers:

```
x-access-token: <JWT_TOKEN>
```

Response:

```
{
  "success": true,
  "message": "Token is valid",
  "user": {
    "id": 1,
    "username": "user@example.com"
  }
}
```

User Management

User Endpoints

Base route: `/api/users`

Available Operations:

- Create user
- Update user
- Delete user
- List users
- Get user by ID

Employee Management

Employee management endpoints handle employee data, shifts, positions, and placements.

Base Routes

- `/api/employees` - Employee CRUD operations
- `/api/positions` - Position management
- `/api/shifts` - Shift management

Common Employee Operations

List Employees

```
GET /api/employees
x-access-token: <JWT_TOKEN>

Query Parameters:
- page (optional): Page number
- limit (optional): Items per page
- search (optional): Search term
```

Get Employee by ID

```
GET /api/employees/:id
x-access-token: <JWT_TOKEN>
```

Create Employee

```
POST /api/employees
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "name": "John Doe",
  "email": "john@example.com",
  "position": "Developer",
  "department": "IT",
```

```
  "phone": "+1234567890",
  "joinDate": "2024-01-01"
}
```

Update Employee

```
PUT /api/employees/:id
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "name": "John Doe Updated",
  "position": "Senior Developer"
}
```

Delete Employee

```
DELETE /api/employees/:id
x-access-token: <JWT_TOKEN>
```

Asset Management

Asset management includes configuration, inventory, and tracking.

Base Routes

- `/api/asset-management/config` - Asset configuration
- `/api/asset-management/inventory` - Inventory management
- `/api/asset-management/tracking` - Asset tracking

Asset Operations

List Assets

```
GET /api/asset-management/inventory
x-access-token: <JWT_TOKEN>
```

Query Parameters:

- status: active, inactive, maintenance
- category: laptop, phone, furniture, etc.
- assignedTo: employee ID

Create Asset

```
POST /api/asset-management/inventory
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "assetCode": "AST-001",
  "name": "MacBook Pro",
  "category": "laptop",
  "serialNumber": "SN123456",
  "purchaseDate": "2024-01-01",
  "value": 2000
}
```

Assign Asset

```
POST /api/asset-management/inventory/:id/assign
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "employeeId": 123,
  "assignDate": "2024-01-15",
  "notes": "Work laptop"
}
```

Track Asset Location

```
GET /api/asset-management/tracking/:assetId
x-access-token: <JWT_TOKEN>
```

Attendance

Attendance system for tracking employee check-ins and check-outs.

Base Routes

- `/api/attendance` - Attendance operations
- `/api/attendance/locations` - Location management

Attendance Operations

Check In

```
POST /api/attendance/check-in
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "employeeId": 123,
  "location": {
    "latitude": -6.2088,
    "longitude": 106.8456
  },
  "photo": "base64_photo_string",
  "timestamp": "2024-01-15T08:00:00Z"
}
```

Check Out

```
POST /api/attendance/check-out
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "employeeId": 123,
  "timestamp": "2024-01-15T17:00:00Z"
}
```

Get Attendance Records

```
GET /api/attendance
x-access-token: <JWT_TOKEN>
```

Query Parameters:

- employeeId: Filter by employee
- startDate: YYYY-MM-DD
- endDate: YYYY-MM-DD
- status: present, absent, late

Attendance Report

GET /api/attendance/report
x-access-token: <JWT_TOKEN>

Query Parameters:

- month: 1-12
- year: YYYY
- departmentId: Filter by department

Leave Management

Leave application and approval system supporting annual, special, and collective leaves.

Base Routes

- `/api/leave` - Leave applications
- `/api/leave/approval` - Approval workflow

Leave Types

- **Annual Leave**: Regular vacation days
- **Special Leave**: Marriage, childbirth, family emergency, etc.
- **Collective Leave**: Company-wide holidays

Leave Operations

Apply for Leave

```
POST /api/leave/apply
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "employeeId": 123,
  "leaveType": "annual",
  "startDate": "2024-02-01",
  "endDate": "2024-02-05",
  "reason": "Family vacation",
  "documents": ["url_to_document"] // Optional
}
```

Get Leave Balance

```
GET /api/leave/balance/:employeeId
x-access-token: <JWT_TOKEN>
```

Response:

```
{
  "success": true,
  "data": {
    "annualLeave": {
      "total": 12,
      "used": 5,
      "remaining": 7
    },
    "specialLeave": {
      "total": 5,
      "used": 1,
      "remaining": 4
    }
  }
}
```

List Leave Applications

```
GET /api/leave
x-access-token: <JWT_TOKEN>
```

Query Parameters:

- status: pending, approved, rejected
- employeeId: Filter by employee
- startDate, endDate: Date range

Approve/Reject Leave

```
POST /api/leave/approval/:leaveId
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "action": "approve", // or "reject"
  "notes": "Approved for vacation"
}
```

Get Leave History

```
GET /api/leave/history/:employeeId
x-access-token: <JWT_TOKEN>
```

Permit Management

Permit system for temporary absence requests (shorter than leave).

Base Routes

- `/api/permit` - Permit applications
- `/api/permit/approval` - Approval workflow

Permit Operations

Apply for Permit

```
POST /api/permit/apply
x-access-token: <JWT_TOKEN>
```

```
Content-Type: application/json

{
  "employeeId": 123,
  "date": "2024-01-15",
  "startTime": "09:00",
  "endTime": "11:00",
  "reason": "Doctor appointment",
  "type": "sick" // or "personal", "official"
}
```

List Permits

```
GET /api/permit
x-access-token: <JWT_TOKEN>

Query Parameters:
- status: pending, approved, rejected
- employeeId: Filter by employee
- date: Specific date
```

Approve/Reject Permit

```
POST /api/permit/approval/:permitId
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "action": "approve",
  "notes": "Medical appointment approved"
}
```

Fleet Management

Driver and vehicle management system.

Base Routes

- `/api/fleet/drivers` - Driver management
- `/api/fleet/auth` - Driver authentication
- `/api/fleet/vehicles` - Vehicle management

Fleet Operations

Register Driver

```
POST /api/fleet/drivers
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "name": "John Driver",
  "licenseNumber": "DL123456",
  "phone": "+1234567890",
  "vehicleType": "truck",
  "licenseExpiry": "2025-12-31"
}
```

Driver Login

```
POST /api/fleet/auth/login
Content-Type: application/json

{
  "username": "driver123",
  "password": "password"
}
```

List Drivers

```
GET /api/fleet/drivers
x-access-token: <JWT_TOKEN>

Query Parameters:
- status: active, inactive
- vehicleType: truck, van, motorcycle
```

Update Driver Location

```
POST /api/fleet/drivers/:id/location
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "latitude": -6.2088,
  "longitude": 106.8456,
  "timestamp": "2024-01-15T10:30:00Z"
}
```

Ticketing

Issue tracking and ticketing system with real-time updates.

Base Routes

- `/api/tickets` - Ticket management
- WebSocket: `/ws/tickets` - Real-time updates

Ticket Operations

Create Ticket

```
POST /api/tickets
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "title": "Network issue in office",
  "description": "WiFi not working on 2nd floor",
  "priority": "high", // low, medium, high, urgent
  "category": "IT",
  "reportedBy": 123,
  "attachments": ["url_to_file"]
}
```

Get Tickets

```
GET /api/tickets
x-access-token: <JWT_TOKEN>

Query Parameters:
- status: open, in_progress, resolved, closed
- priority: low, medium, high, urgent
- assignedTo: employee ID
- category: IT, HR, Admin, etc.
```

Update Ticket

```
PUT /api/tickets/:id
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "status": "in_progress",
  "assignedTo": 456,
  "notes": "Working on the issue"
}
```

Add Comment

```
POST /api/tickets/:id/comments
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "comment": "Issue resolved by resetting the router",
  "internal": false // true for internal notes
}
```

Close Ticket

```
POST /api/tickets/:id/close
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "resolution": "Reset router and reconfigured network",
```

```
  "satisfaction": 5 // 1-5 rating
}
```

Meeting Room

Meeting room booking and scheduling system.

Base Routes

- `/api/meeting-rooms` - Room management
- `/api/meeting-rooms/schedule` - Schedule management

Meeting Room Operations

List Meeting Rooms

```
GET /api/meeting-rooms
x-access-token: <JWT_TOKEN>

Query Parameters:
- capacity: Minimum capacity
- floor: Floor number
- facilities: projector, whiteboard, video_conference
```

Book Meeting Room

```
POST /api/meeting-rooms/schedule
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "roomId": 5,
  "title": "Team Standup",
  "date": "2024-01-15",
  "startTime": "09:00",
  "endTime": "10:00",
  "attendees": [123, 456, 789],
```

```
  "description": "Daily team standup meeting"
}
```

Check Room Availability

```
GET /api/meeting-rooms/:id/availability
x-access-token: <JWT_TOKEN>

Query Parameters:
- date: YYYY-MM-DD
```

Cancel Booking

```
DELETE /api/meeting-rooms/schedule/:id
x-access-token: <JWT_TOKEN>
```

Branding Approval

Approval workflow for branding materials with real-time updates.

Base Routes

- `/api/branding-approval` - Approval management
- WebSocket: `/ws/branding-approval` - Real-time updates

Branding Approval Operations

Submit for Approval

```
POST /api/branding-approval
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "title": "New Logo Design",
  "description": "Updated company logo",
  "category": "logo",
```

```
  "files": ["url_to_design_file"],
  "requestedBy": 123,
  "approvers": [456, 789]
}
```

List Approval Requests

```
GET /api/branding-approval
x-access-token: <JWT_TOKEN>

Query Parameters:
- status: pending, approved, rejected, revision
- requestedBy: employee ID
- category: logo, marketing, social_media
```

Approve/Reject

```
POST /api/branding-approval/:id/review
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "action": "approve", // approve, reject, request_revision
  "feedback": "Looks great, approved!",
  "revisionNotes": "" // Required if action is request_revision
}
```

CMS (Content Management System)

Content management for blog, ads, training, careers, and more.

Base Routes

- `/api/cms/blog` - Blog posts
- `/api/cms/ads` - Advertisements

- `/api/cms/training` - Training materials
- `/api/cms/career` - Career/job postings
- `/api/cms/vacancies` - Vacancy listings
- `/api/cms/department` - Department information
- `/api/cms/droppoint` - Drop point locations

Blog Operations

Create Blog Post

```
POST /api/cms/blog
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "title": "Company Update 2024",
  "slug": "company-update-2024",
  "content": "Full blog post content here...",
  "excerpt": "Short description",
  "featuredImage": "url_to_image",
  "author": 123,
  "categories": ["company-news", "updates"],
  "tags": ["2024", "announcement"],
  "status": "draft" // draft, published
}
```

List Blog Posts

```
GET /api/cms/blog
x-access-token: <JWT_TOKEN>

Query Parameters:
- status: draft, published
- author: author ID
- category: category slug
- search: search term
```

Update Blog Post

```
PUT /api/cms/blog/:id
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "title": "Updated Title",
  "status": "published"
}
```

Advertisement Operations

Create Ad

```
POST /api/cms/ads
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "title": "Summer Sale",
  "image": "url_to_image",
  "link": "https://sale.example.com",
  "placement": "homepage_banner", // homepage_banner, sidebar, popup
  "startDate": "2024-06-01",
  "endDate": "2024-08-31",
  "active": true
}
```

Career/Vacancy Operations

Create Job Posting

```
POST /api/cms/vacancies
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "title": "Senior Software Engineer",
  "department": "Engineering",
```

```
  "location": "Jakarta, Indonesia",
  "type": "full-time", // full-time, part-time, contract
  "description": "Job description...",
  "requirements": ["5+ years experience", "Node.js expertise"],
  "salary": "Competitive",
  "closingDate": "2024-02-28",
  "status": "open"
}
```

File Upload

File upload and management endpoints.

Upload Temporary File

```
POST /api/upload/temp
x-access-token: <JWT_TOKEN>
Content-Type: multipart/form-data

Body: file (form data)
```

Response:

```
{
  "success": true,
  "name": "1234567890-filename.pdf",
  "url": "http://localhost:3601/api/temp/1234567890-filename.pdf",
  "status": "done"
}
```

Download Temporary File

```
GET /api/temp/:filename

Query Parameters:
- s=download (optional): Force download instead of display
```

Note: Temporary files are automatically deleted after first download or after 24 hours.

Download Static File

```
GET /api/static/:filename
```

Sales

Sales tracking and management.

Base Routes

- `/api/sales` - Sales operations

Sales Operations

Create Sales Record

```
POST /api/sales
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "customerId": 123,
  "items": [
    {
      "productId": 456,
      "quantity": 2,
      "price": 100
    }
  ],
  "total": 200,
  "salesPerson": 789,
  "date": "2024-01-15"
}
```

Get Sales Report

```
GET /api/sales/report
x-access-token: <JWT_TOKEN>

Query Parameters:
- startDate: YYYY-MM-DD
- endDate: YYYY-MM-DD
- salesPerson: employee ID
- customerId: customer ID
```

Freelance

Freelancer management system.

Base Routes

- `/api/freelance` - Freelancer operations

Freelance Operations

Register Freelancer

```
POST /api/freelance
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "name": "Jane Freelancer",
  "email": "jane@example.com",
  "skills": ["design", "video editing"],
  "hourlyRate": 50,
  "portfolio": "https://portfolio.example.com"
}
```

Assign Project

```
POST /api/freelance/:id/projects
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "projectName": "Website Redesign",
  "description": "Redesign company website",
  "deadline": "2024-03-01",
  "budget": 5000
}
```

Onapps

Application-specific features and voucher management.

Base Routes

- `/api/onapps` - General onapps operations
- `/api/onapps/vouchers` - Voucher management

Voucher Operations

Create Voucher

```
POST /api/onapps/vouchers
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "code": "SAVE20",
  "description": "20% off all orders",
  "discountType": "percentage", // percentage or fixed
  "discountValue": 20,
  "minPurchase": 100,
  "maxDiscount": 50,
  "validFrom": "2024-01-01",
```

```
  "validUntil": "2024-12-31",
  "usageLimit": 1000,
  "active": true
}
```

Validate Voucher

```
POST /api/onapps/vouchers/validate
x-access-token: <JWT_TOKEN>
Content-Type: application/json

{
  "code": "SAVE20",
  "orderAmount": 150
}
```

Response:

```
{
  "success": true,
  "valid": true,
  "discount": 30,
  "finalAmount": 120,
  "message": "Voucher applied successfully"
}
```

WebSocket Endpoints

Branding Approval WebSocket

Connection:

```
const ws = new WebSocket('ws://localhost:3601/ws/branding-approval');

ws.onopen = () => {
  console.log('Connected to branding approval updates');
};
```

```
ws.onmessage = (event) => {
  const data = JSON.parse(event.data);
  console.log('Approval update:', data);
};
```

Message Format:

```
{
  "type": "approval_update",
  "data": {
    "requestId": 123,
    "status": "approved",
    "approver": "John Doe",
    "timestamp": "2024-01-15T10:30:00Z"
  }
}
```

Ticketing WebSocket

Connection:

```
const ws = new WebSocket('ws://localhost:3601/ws/tickets');

ws.onmessage = (event) => {
  const data = JSON.parse(event.data);
  // Handle ticket updates
};
```

Rate Limiting

(Future implementation)

API rate limits:

- 100 requests per 15 minutes per IP address
- 1000 requests per hour for authenticated users

Deprecation Notice

(None currently)

API Version: 1.0.0
Last Updated: 2026-02-04

For additional support or questions about the API, please contact the development team.

Architecture

This document provides a detailed overview of the ON Internal API system architecture, data flows, and technical design decisions.

New Page (will be separated)

Table of Contents

- System Overview
- Architecture Layers
- Database Architecture
- Authentication & Authorization
- API Design
- WebSocket Architecture
- File Management
- Scheduled Tasks
- Security Architecture
- Scalability Considerations

System Overview

ON Internal API is a multi-tenant, microservices-style backend system built on Node.js and Express.js. It manages multiple business domains through a unified API gateway while maintaining separation of concerns through modular architecture.

Design Principles

1. **Modularity**: Each business domain (employee, asset, fleet, etc.) is isolated in its own module
2. **Separation of Concerns**: Clear separation between routes, controllers, models, and utilities
3. **Database Segregation**: Different databases for different domains to ensure data isolation
4. **Security First**: Multiple layers of security (Helmet, CORS, JWT, validation)
5. **Real-time Capable**: WebSocket support for live updates

6. **Stateless**: JWT-based authentication allows horizontal scaling

Architecture Layers

1. Entry Point Layer (server.js)

```
┌─────────────────────────────────────────────────────┐
│                   server.js                         │
├─────────────────────────────────────────────────────┤
│  - Load environment variables (dotenv)              │
│  - Initialize Express application                   │
│  - Configure security middleware (Helmet, CORS)     │
│  - Setup logging (Morgan with colored output)       │
│  - Connect to all databases (Sequelize/Mongoose)    │
│  - Create directory structure                       │
│  - Load all routes                                  │
│  - Initialize WebSocket server                      │
│  - Start HTTP server                                │
│  - Load scheduled tasks                             │
└─────────────────────────────────────────────────────┘
```

Key Responsibilities:

- Application bootstrap
- Global configuration
- Middleware stack setup
- Database connection management
- Route registration
- Server initialization

2. Middleware Layer

```
┌─────────────────────────────────────────────────────┐
│              Middleware Pipeline                    │
├─────────────────────────────────────────────────────┤
│  1. CORS (Cross-Origin Resource Sharing)            │
│  2. Helmet (Security headers)                       │
```

```
│ 3. Morgan (Request logging with colors)          │
│ 4. Express JSON/URL-encoded parsers (10MB limit) │
│ 5. Custom middleware (per route):                │
│    - authJwt.verifyToken (JWT validation)        │
│    - contentTypeValid (Content-Type check)       │
│    - validator.validate (Schema validation)      │
│    - checkEmployee/checkFleet (Role validation)  │
│ 6. Error handler (errorParamHandler)             │
└──────────────────────────────────────────────────┘
```

Middleware Components:

Security Middleware (`app/middleware/authJwt.js`):

```
verifyToken(req, res, next) {
  // Extract token from header
  // Verify JWT signature
  // Decode and attach user to request
  // Continue or reject
}
```

Validation Middleware (`app/schemas/`):

- Uses AJV for JSON Schema validation
- Validates request body, query params, and URL params
- Returns structured error messages

Error Handler (`app/middleware/errorParamHandler.js`):

- Catches all errors
- Formats error responses
- Logs errors
- Returns consistent JSON error format

3. Route Layer

Routes define API endpoints and map them to controllers:

```
// Example: app/routes/auth.routes.js
module.exports = function (app) {
  app.post(
    "/api/auth/login",
```

```
    [
      contentTypeValid("application/json"),
      validator.validate({ body: auth.login }),
    ],
    authController.login
  );

  app.get(
    "/api/auth/account",
    authJwt.verifyToken,
    authController.loadAccount
  );
};
```

Route Organization:

- One route file per module
- Middleware applied declaratively
- Clear HTTP method usage
- RESTful design patterns

4. Controller Layer

Controllers contain business logic:

```
┌─────────────────────────────────────────────────┐
│            Controller Responsibilities          │
├─────────────────────────────────────────────────┤
│  1. Extract data from request (body, query, params) │
│  2. Call database operations via models         │
│  3. Process business logic                      │
│  4. Format response data                        │
│  5. Handle errors                               │
│  6. Return HTTP response                        │
└─────────────────────────────────────────────────┘
```

Controller Pattern:

```
exports.methodName = async (req, res) => {
  try {
```

```
    // 1. Extract data
    const { param } = req.body;

    // 2. Validate business rules
    if (!param) throw new Error("Missing parameter");

    // 3. Database operations
    const result = await Model.findOne({ where: { id } });

    // 4. Process data
    const processed = processData(result);

    // 5. Return response
    res.status(200).json({
      success: true,
      data: processed
    });
  } catch (error) {
    // 6. Error handling
    res.status(400).json({
      success: false,
      message: error.message
    });
  }
};
```

5. Model Layer

Models define database schemas using Sequelize (PostgreSQL) or Mongoose (MongoDB):

```
// Example Sequelize model
module.exports = (sequelize, Sequelize) => {
  const Model = sequelize.define("table_name", {
    id: {
      type: Sequelize.INTEGER,
      primaryKey: true,
      autoIncrement: true
    },
```

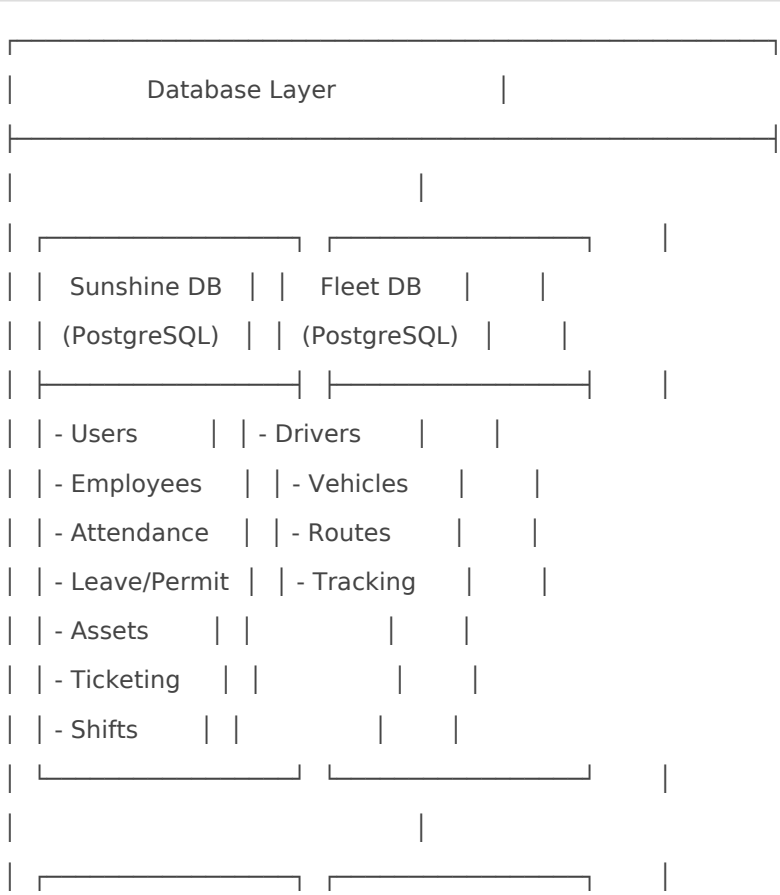
```
    name: {
      type: Sequelize.STRING,
      allowNull: false
    },
    // ... other fields
  }, {
    timestamps: true,
    tableName: "table_name"
  });

  return Model;
};
```

Database Architecture

Multi-Database Strategy

The system uses four separate databases:



```
│ │    CMS DB      │ │    Mitra DB     │       │
│ │ (PostgreSQL)   │ │ (PostgreSQL +   │       │
│ ├────────────────┤ │    MongoDB)     │       │
│ │ - Blog Posts   │ ├─────────────────┤       │
│ │ - Ads          │ │ - Mitra data    │       │
│ │ - Training     │ │ - Documents     │       │
│ │ - Career       │ │                 │       │
│ │ - Departments  │ │                 │       │
│ │ - Drop Points  │ │                 │       │
│ └────────────────┘ └─────────────────┘       │
└─────────────────────────────────────────────────┘
```

Database Configuration

Connection Pooling (for performance):

```
pool: {
  max: 5,      // Maximum connections
  min: 0,      // Minimum connections
  acquire: 30000,  // Max time to get connection (ms)
  idle: 10000  // Max idle time before closing (ms)
}
```

Database Selection Pattern

```
// In models/index.js
const sequelizeSunshine = new Sequelize(sunshineDB.DB, ...);
const sequelizeFleet = new Sequelize(fleetDB.DB, ...);
const sequelizeCms = new Sequelize(cmsDB.DB, ...);

// Models are registered to specific connections
db.employee = require("./employee.model")(sequelizeSunshine, Sequelize);
db.driver = require("./driver.model")(sequelizeFleet, Sequelize);
```

Data Relationships

Cross-Database Relationships:

- Avoided when possible for better independence
- When needed, handled at application layer (not DB constraints)
- Use foreign keys within same database

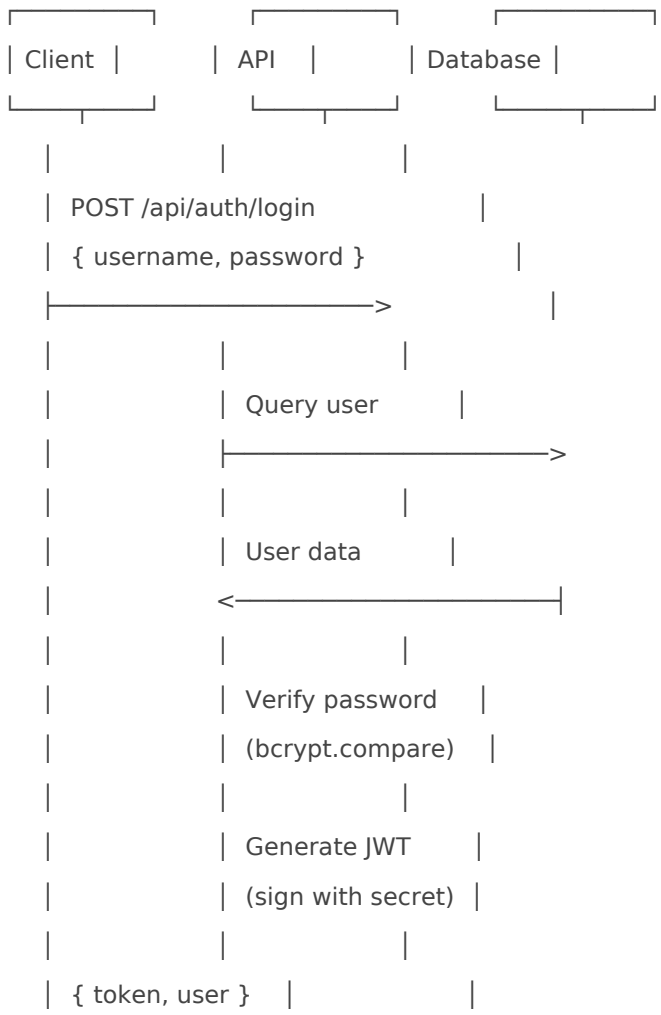
Example:

```
// Employee (Sunshine DB) -> Asset Assignment (Sunshine DB) ✓
// Employee has Assets relationship

// Employee (Sunshine DB) -> Driver (Fleet DB) ✗
// Handled via employee_id field and application logic
```

Authentication & Authorization

JWT Authentication Flow





Authentication Components

Login Process (`app/controllers/auth.controller.js`):

1. Receive username/password
2. Query database for user
3. Verify password with bcrypt
4. Generate JWT token with user data
5. Return token and user info

Token Verification (`app/middleware/authJwt.js`):

1. Extract token from header (`x-access-token`)
2. Verify token signature with JWT_SECRET
3. Decode payload (user ID, roles, etc.)
4. Attach decoded data to `req.user`
5. Continue to next middleware/controller

Token Structure:

```
{
  id: userId,
  username: "user@example.com",
  role: "admin",
  iat: 1234567890,  // Issued at
  exp: 1234654290   // Expires at
}
```

Authorization Patterns

Role-Based Access Control:

```
// In middleware
if (req.user.role !== 'admin') {
  return res.status(403).json({
    message: "Unauthorized access"
  });
}
```

Resource-Based Authorization:

```
// Verify user owns resource
const resource = await Model.findOne({
  where: { id: resourceId, userId: req.user.id }
});

if (!resource) {
  return res.status(403).json({
    message: "Access denied"
  });
}
```

API Design

RESTful Principles

Resource Naming:

- Plural nouns: `/api/employees`, `/api/assets`
- Nested resources: `/api/employees/:id/shifts`
- Actions as sub-resources: `/api/tickets/:id/assign`

HTTP Methods:

- `GET`: Read/retrieve data
- `POST`: Create new resource
- `PUT`: Update entire resource
- `PATCH`: Partial update
- `DELETE`: Remove resource

Status Codes:

- `200`: Success
- `201`: Created
- `400`: Bad request (validation error)
- `401`: Unauthorized (not authenticated)
- `403`: Forbidden (not authorized)
- `404`: Not found
- `500`: Internal server error

Response Format

Success Response:

```
{
  "success": true,
  "data": {
    "id": 1,
    "name": "Example"
  },
  "message": "Operation successful"
}
```

Error Response:

```
{
  "success": false,
  "message": "Validation failed",
  "errors": [
    {
      "field": "email",
```

```
      "message": "Invalid email format"
    }
  ]
}
```

Pagination (when applicable):

```
{
  "success": true,
  "data": [...],
  "pagination": {
    "page": 1,
    "perPage": 20,
    "total": 150,
    "totalPages": 8
  }
}
```

Input Validation

Using AJV JSON Schema:

```
// app/schemas/example.schema.js
module.exports = {
  create: {
    type: "object",
    properties: {
      name: { type: "string", minLength: 1 },
      email: { type: "string", format: "email" },
      age: { type: "integer", minimum: 18 }
    },
    required: ["name", "email"],
    additionalProperties: false
  }
};
```

Applied in routes:

```
app.post(
  "/api/example",
  validator.validate({ body: exampleSchema.create }),
  controller.create
);
```

WebSocket Architecture

WebSocket Server Setup

```
// app/routes/ws.routes.js
const WebSocket = require("ws");

module.exports = (server, app) => {
  const wss = new WebSocket.Server({ noServer: true });

  server.on("upgrade", (request, socket, head) => {
    // Parse URL to route to correct WebSocket handler
    const pathname = url.parse(request.url).pathname;

    if (pathname === "/ws/branding-approval") {
      wss.handleUpgrade(request, socket, head, (ws) => {
        wss.emit("connection", ws, request);
      });
    }
  });

  wss.on("connection", (ws, request) => {
    // Handle connection
    ws.on("message", (message) => {
      // Handle message
    });

    ws.on("close", () => {
      // Handle disconnect
    });
```

```
  });
};
```

WebSocket Endpoints

1. **Branding Approval** (`/ws/branding-approval`)
   - Real-time approval status updates
   - Mobile and web clients
   - Notification delivery
2. **Ticketing** (`/ws/ticketing`)
   - Ticket status updates
   - Assignment notifications
   - Real-time chat (if applicable)

Connection Management

Heartbeat (Ping/Pong):

```
// interval.js
setInterval(function ping() {
  wss.clients.forEach(function each(ws) {
    if (ws.isAlive === false) {
      return ws.terminate();
    }
    ws.isAlive = false;
    ws.ping();
  });
}, 30000);  // Every 30 seconds
```

Client Tracking:

```
// Track connected clients
const clients = new Map();

wss.on("connection", (ws, request) => {
  const userId = extractUserId(request);
  clients.set(userId, ws);

  ws.on("close", () => {
    clients.delete(userId);
```

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

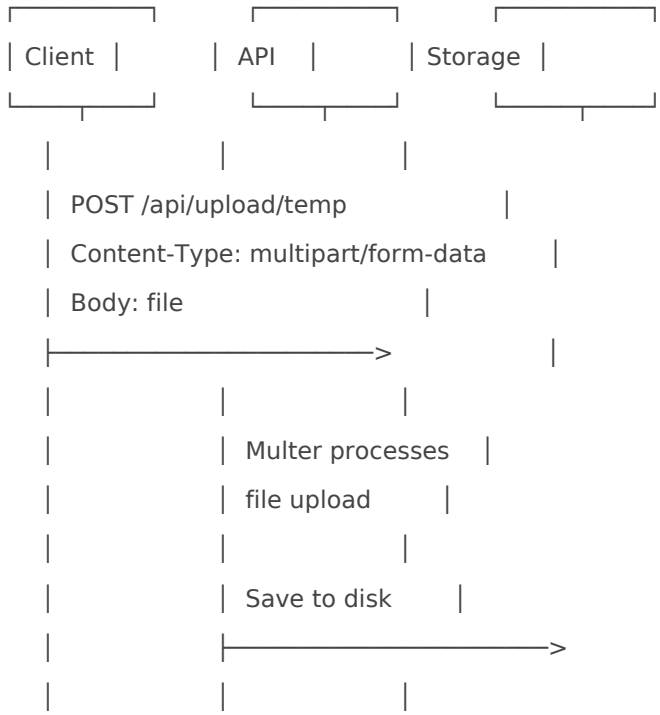
Broadcasting:

```
// Send to all clients
wss.clients.forEach((client) => {
  if (client.readyState === WebSocket.OPEN) {
    client.send(JSON.stringify(data));
  }
});

// Send to specific user
const userWs = clients.get(userId);
if (userWs && userWs.readyState === WebSocket.OPEN) {
  userWs.send(JSON.stringify(data));
}
```

File Management

Upload Flow



```
    |                |  File saved          |
    |                <──────────────────────┤
    |                |                      |
    |  { url, name }      |                 |
    <──────────────────────┤                 |
    |                |                      |
```

Storage Structure

```
resources/
├── static/              # Permanent files
│   ├── employee/        # Employee photos, documents
│   ├── ticketing/       # Ticket attachments
│   ├── cms/
│   │   ├── ad/          # Advertisement images
│   │   ├── blog/        # Blog post images
│   │   ├── training/    # Training materials
│   │   └── department/  # Department logos
│   ├── branding-approval/  # Branding assets
│   └── onapps/
│       └── voucher/     # Voucher images
└── temp/                # Temporary files (auto-deleted)
```

File Upload Middleware

```
// app/middleware/uploadTemp.js
const multer = require("multer");
const path = require("path");

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    cb(null, path.join(__remoteDir, __stageEnv, "temp"));
  },
  filename: (req, file, cb) => {
    const uniqueName = `${Date.now()}-${file.originalname}`;
    cb(null, uniqueName);
  }
```

```
});

const upload = multer({
  storage: storage,
  limits: { fileSize: 10 * 1024 * 1024 },  // 10MB
  fileFilter: (req, file, cb) => {
    // Validate file type if needed
    cb(null, true);
  }
});
```

File Serving

Temporary Files:

- Served via `/api/temp/:filename`
- Automatically deleted after first download
- Used for reports, exports, etc.

Static Files:

- Served via `/api/static/:filename`
- Permanent storage
- Used for user uploads, images, etc.

Scheduled Tasks

Task Scheduler Architecture

```
// app/schedulers/file-temporary.scheduler.js
const cron = require("node-cron");

exports.cleanTempFile = () => {
  // Run every hour
  cron.schedule("0 * * * *", async () => {
    const tempDir = path.join(__remoteDir, __stageEnv, "temp");
    const files = fs.readdirSync(tempDir);
```

```
    files.forEach(file => {
      const filePath = path.join(tempDir, file);
      const stats = fs.statSync(filePath);
      const now = Date.now();
      const age = now - stats.mtimeMs;

      // Delete files older than 24 hours
      if (age > 24 * 60 * 60 * 1000) {
        fs.unlinkSync(filePath);
      }
    });
  });
};
```

Scheduled Tasks

1. **Temporary File Cleanup**
   - Schedule: Every hour
   - Action: Delete temp files older than 24 hours
   - Purpose: Prevent disk space issues
2. **Leave Auto-Rejection**
   - Schedule: Daily
   - Action: Auto-reject pending leave requests past deadline
   - Purpose: Workflow automation

Adding New Scheduled Tasks

```
// 1. Create scheduler file
// app/schedulers/your-task.scheduler.js
const cron = require("node-cron");

exports.yourTask = () => {
  cron.schedule("0 0 * * *", async () => {  // Daily at midnight
    // Your task logic
  });
};

// 2. Load in server.js
```

```
const yourTaskScheduler = require("./app/schedulers/your-task.scheduler");
yourTaskScheduler.yourTask();
```

Security Architecture

Security Layers

```
┌─────────────────────────────────────────────────┐
│           Security Layers                       │
├─────────────────────────────────────────────────┤
│ 1. Transport Layer (HTTPS in production)        │
│ 2. CORS (Cross-Origin Resource Sharing)         │
│ 3. Helmet.js (Security headers)                 │
│ 4. Request Size Limits (10MB)                   │
│ 5. JWT Authentication                           │
│ 6. Input Validation (AJV JSON Schema)           │
│ 7. SQL Injection Protection (Sequelize ORM)     │
│ 8. Password Hashing (bcryptjs)                  │
│ 9. Error Sanitization (no stack traces in prod) │
└─────────────────────────────────────────────────┘
```

Security Headers (Helmet.js)

Automatically applied headers:

- `X-DNS-Prefetch-Control`
- `X-Frame-Options`
- `X-Content-Type-Options`
- `X-XSS-Protection`
- `Strict-Transport-Security` (HTTPS only)

CORS Configuration

```
app.use(cors());  // Allow all origins (configure for production)


// Custom CORS for specific resources
```

```
app.use((req, res, next) => {
  res.setHeader("Cross-Origin-Resource-Policy", "cross-origin");
  next();
});
```

Production CORS (recommended):

```
const corsOptions = {
  origin: [
    "https://yourdomain.com",
    "https://app.yourdomain.com"
  ],
  credentials: true
};
app.use(cors(corsOptions));
```

SQL Injection Prevention

Using Sequelize ORM with parameterized queries:

```
// ✓ Safe - parameterized
User.findOne({
  where: { email: userInput }
});

// ✗ Unsafe - raw query without parameters
sequelize.query(`SELECT * FROM users WHERE email = '${userInput}'`);

// ✓ Safe - raw query with parameters
sequelize.query(
  "SELECT * FROM users WHERE email = :email",
  {
    replacements: { email: userInput },
    type: QueryTypes.SELECT
  }
);
```

Password Security

```
const bcrypt = require("bcryptjs");

// Hashing (on user creation)
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(password, salt);

// Verification (on login)
const isValid = await bcrypt.compare(password, user.password);
```

Scalability Considerations

Horizontal Scaling

Stateless Design:

- JWT tokens (no server-side sessions)
- No in-memory state (use database/cache)
- Load balancer compatible

Database Connection Pooling:

- Limited connections per instance
- Reuse existing connections
- Automatic cleanup of idle connections

Performance Optimization

Database Queries:

- Use indexes on frequently queried fields
- Limit result sets (pagination)
- Use `select` to fetch only needed fields
- Avoid N+1 queries (use `include` for joins)

Caching (future consideration):

```
// Example with Redis
const redis = require("redis");
const client = redis.createClient();
```

```
// Cache frequently accessed data
const cached = await client.get(key);
if (cached) return JSON.parse(cached);

const data = await database.query();
await client.setex(key, 3600, JSON.stringify(data));
```

Request Rate Limiting (future consideration):

```
const rateLimit = require("express-rate-limit");

const limiter = rateLimit({
  windowMs: 15 * 60 * 1000,  // 15 minutes
  max: 100  // Limit each IP to 100 requests per windowMs
});

app.use("/api/", limiter);
```

Monitoring & Logging

Current Logging:

- Morgan for HTTP request logging
- Colored console output for readability
- Sequelize query logging

Production Logging (recommended):

- Use Winston or Bunyan
- Log to files/external service
- Structured logging (JSON format)
- Different log levels (error, warn, info, debug)

Metrics (future consideration):

- Response times
- Error rates
- Database query performance
- Active connections
- Memory/CPU usage

Document Version: 1.0.0
Last Updated: 2026-02-04

Deployment

Complete guide for deploying the ON Internal API to production environments.

New Page

Table of Contents

- Prerequisites
- Server Requirements
- Pre-Deployment Checklist
- Deployment Methods
- Production Configuration
- Database Setup
- Environment Variables
- Running in Production
- Reverse Proxy Setup
- SSL/TLS Configuration
- Monitoring
- Backup Strategy
- Troubleshooting
- Rollback Procedures
- Performance Tuning

Prerequisites

Before deploying to production, ensure you have:

- [ ] Server with minimum specifications (see Server Requirements)
- [ ] Domain name configured
- [ ] SSL/TLS certificate
- [ ] PostgreSQL database server
- [ ] MongoDB server (if using Mitra MongoDB features)

- [ ] Firebase project (if using Firebase features)
- [ ] Backup solution
- [ ] Monitoring tools

Server Requirements

Minimum Specifications

Single Server Setup:

- **CPU**: 2 cores (4 cores recommended)
- **RAM**: 4GB (8GB recommended)
- **Storage**: 50GB SSD
- **OS**: Ubuntu 20.04 LTS or higher, CentOS 8+, or similar Linux distribution
- **Network**: Static IP address, ports 80/443 open

Multi-Server Setup (Recommended for Production):

- **Application Server**: 2-4 cores, 4-8GB RAM
- **Database Server**: 4-8 cores, 16-32GB RAM
- **Load Balancer**: 2 cores, 4GB RAM (if using multiple app servers)

Software Requirements

- **Node.js**: v18.x or v20.x LTS
- **npm**: v9.x or higher
- **PostgreSQL**: v12.x or higher
- **MongoDB**: v4.4 or higher (optional)
- **Nginx** or **Apache**: For reverse proxy
- **PM2** or **systemd**: For process management
- **Git**: For code deployment

Pre-Deployment Checklist

Code Preparation

- [ ] All tests passing
- [ ] Code reviewed and approved
- [ ] No console.log statements in production code
- [ ] All dependencies updated and audited (`npm audit`)
- [ ] Environment variables documented
- [ ] Database migrations prepared
- [ ] API documentation updated

Security Checklist

- [ ] JWT_SECRET is strong and unique
- [ ] Database passwords are strong
- [ ] CORS configured for production domains only
- [ ] Helmet.js security headers configured
- [ ] Rate limiting implemented (if required)
- [ ] Input validation enabled on all endpoints
- [ ] File upload size limits set
- [ ] SQL injection prevention verified
- [ ] XSS prevention verified

Infrastructure Checklist

- [ ] Database backups configured
- [ ] Log rotation configured
- [ ] Monitoring set up
- [ ] Alerting configured
- [ ] SSL certificate obtained and installed
- [ ] Firewall rules configured
- [ ] DNS records configured

Deployment Methods

Method 1: Manual Deployment

1. Server Setup

```
# Update system
sudo apt update && sudo apt upgrade -y

# Install Node.js (v20.x)
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash -
sudo apt install -y nodejs

# Install PM2 globally
sudo npm install -g pm2

# Install PostgreSQL
sudo apt install -y postgresql postgresql-contrib

# Install Nginx
sudo apt install -y nginx

# Install Git
sudo apt install -y git
```

2. Create Application User

```
# Create user for running the application
sudo useradd -m -s /bin/bash ondelivery
sudo su - ondelivery
```

3. Clone Repository

```
# Clone the application
git clone <repository-url> /home/ondelivery/on-internal-api
cd /home/ondelivery/on-internal-api

# Checkout production branch
git checkout main  # or production branch
```

4. Install Dependencies

```
# Install production dependencies only
NODE_ENV=production npm ci
```

5. Configure Environment

```
# Create .env file
nano .env

# Add production environment variables (see Environment Variables section)
```

6. Set Permissions

```
# Ensure proper permissions
chmod 755 /home/ondelivery/on-internal-api
chmod 600 /home/ondelivery/on-internal-api/.env

# Create resources directories
mkdir -p resources/static/{employee,ticketing,cms,branding-approval,onapps}
mkdir -p resources/temp
```

Method 2: Docker Deployment

Docker Setup

Dockerfile:

```
FROM node:20-alpine

# Create app directory
WORKDIR /usr/src/app

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

# Copy app source
COPY . .

# Create directories
```

```
RUN mkdir -p resources/static resources/temp

# Expose port
EXPOSE 3601

# Start application
CMD ["node", "server.js"]
```

docker-compose.yml:

```
version: '3.8'

services:
  app:
    build: .
    ports:
      - "3601:3601"
    environment:
      - NODE_ENV=production
    env_file:
      - .env
    volumes:
      - ./resources:/usr/src/app/resources
    depends_on:
      - postgres
    restart: unless-stopped

  postgres:
    image: postgres:14
    environment:
      POSTGRES_USER: ${PG_SUNSHINE_USER}
      POSTGRES_PASSWORD: ${PG_SUNSHINE_PASSWORD}
      POSTGRES_DB: ${PG_SUNSHINE_DB}
    volumes:
      - postgres_data:/var/lib/postgresql/data
    restart: unless-stopped

volumes:
  postgres_data:
```

Deploy with Docker:

```
# Build and start
docker-compose up -d

# View logs
docker-compose logs -f app

# Stop
docker-compose down
```

Method 3: CI/CD Pipeline

GitHub Actions Example:

```
# .github/workflows/deploy.yml
name: Deploy to Production

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Deploy to Server
        uses: appleboy/ssh-action@master
        with:
          host: ${{ secrets.SERVER_HOST }}
          username: ${{ secrets.SERVER_USER }}
          key: ${{ secrets.SSH_PRIVATE_KEY }}
          script: |
            cd /home/ondelivery/on-internal-api
            git pull origin main
            npm ci --only=production
```

```
pm2 restart on-internal-api
```

Production Configuration

PM2 Configuration

ecosystem.config.js:

```
module.exports = {
  apps: [{
    name: 'on-internal-api',
    script: './server.js',
    instances: 'max',  // Use all available CPU cores
    exec_mode: 'cluster',
    env: {
      NODE_ENV: 'production',
      PORT: 3601
    },
    error_file: './logs/err.log',
    out_file: './logs/out.log',
    log_date_format: 'YYYY-MM-DD HH:mm:ss Z',
    merge_logs: true,
    max_memory_restart: '1G',
    autorestart: true,
    watch: false
  }]
};
```

Start with PM2:

```
# Start application
pm2 start ecosystem.config.js

# Save PM2 process list
pm2 save

# Setup PM2 to start on boot
```

```
pm2 startup
# Follow the instructions provided

# Monitor application
pm2 monit

# View logs
pm2 logs on-internal-api
```

Systemd Service (Alternative to PM2)

Create service file:

```
sudo nano /etc/systemd/system/on-internal-api.service
```

Service configuration:

```
[Unit]
Description=ON Internal API
After=network.target postgresql.service

[Service]
Type=simple
User=ondelivery
WorkingDirectory=/home/ondelivery/on-internal-api
ExecStart=/usr/bin/node /home/ondelivery/on-internal-api/server.js
Restart=on-failure
RestartSec=10
StandardOutput=syslog
StandardError=syslog
SyslogIdentifier=on-internal-api
Environment=NODE_ENV=production

[Install]
WantedBy=multi-user.target
```

Enable and start service:

```
sudo systemctl daemon-reload
sudo systemctl enable on-internal-api
sudo systemctl start on-internal-api
sudo systemctl status on-internal-api
```

Database Setup

PostgreSQL Configuration

1. Create Databases

```
-- Connect as postgres user
sudo -u postgres psql

-- Create databases
CREATE DATABASE sunshine_db;
CREATE DATABASE fleet_db;
CREATE DATABASE cms_db;
CREATE DATABASE mitra_db;

-- Create user
CREATE USER ondelivery WITH PASSWORD 'strong_password_here';

-- Grant privileges
GRANT ALL PRIVILEGES ON DATABASE sunshine_db TO ondelivery;
GRANT ALL PRIVILEGES ON DATABASE fleet_db TO ondelivery;
GRANT ALL PRIVILEGES ON DATABASE cms_db TO ondelivery;
GRANT ALL PRIVILEGES ON DATABASE mitra_db TO ondelivery;

\q
```

2. Run Migrations

```
cd /home/ondelivery/on-internal-api

# If using migrations
npm run migrate
```

```
# Or sync models (development approach)
# Uncomment sync commands in server.js temporarily
node server.js
# Then ctrl+C and comment them back out
```

3. PostgreSQL Performance Tuning

Edit `/etc/postgresql/14/main/postgresql.conf`:

```
# Memory settings (for 16GB RAM server)
shared_buffers = 4GB
effective_cache_size = 12GB
maintenance_work_mem = 1GB
work_mem = 32MB

# Connection settings
max_connections = 200

# Query optimization
random_page_cost = 1.1  # For SSD
effective_io_concurrency = 200

# Write-ahead log
wal_buffers = 16MB
min_wal_size = 1GB
max_wal_size = 4GB

# Checkpoints
checkpoint_completion_target = 0.9
```

Restart PostgreSQL:

```
sudo systemctl restart postgresql
```

MongoDB Configuration (If Used)

```
# Install MongoDB
wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -
```

```
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" |
sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list
sudo apt update
sudo apt install -y mongodb-org

# Start MongoDB
sudo systemctl start mongod
sudo systemctl enable mongod

# Secure MongoDB
mongo
use admin
db.createUser({
  user: "ondelivery",
  pwd: "strong_password",
  roles: ["readWriteAnyDatabase"]
})
exit

# Enable authentication in /etc/mongod.conf
sudo nano /etc/mongod.conf
```

Add:

```
security:
  authorization: enabled
```

Restart:

```
sudo systemctl restart mongod
```

Environment Variables

Production .env Template

```
# Server Configuration
NODE_ENV=production
```

```
SERVER_PORT=3601
BASE_DOMAIN=https://api.yourdomain.com
DIR=/home/ondelivery/resources

# PostgreSQL - Sunshine Database
PG_SUNSHINE_HOST=localhost
PG_SUNSHINE_PORT=5432
PG_SUNSHINE_USER=ondelivery
PG_SUNSHINE_PASSWORD=your_strong_password
PG_SUNSHINE_DB=sunshine_db
PG_SUNSHINE_DIALECT=postgres
PG_SUNSHINE_MAX_POOL=20
PG_SUNSHINE_MIN_POOL=5
PG_SUNSHINE_ACQUIRE_POOL=30000
PG_SUNSHINE_IDLE_POOL=10000

# PostgreSQL - Fleet Database
PG_FLEET_HOST=localhost
PG_FLEET_PORT=5432
PG_FLEET_USER=ondelivery
PG_FLEET_PASSWORD=your_strong_password
PG_FLEET_DB=fleet_db
PG_FLEET_dialect=postgres
PG_FLEET_MAX_POOL=10
PG_FLEET_MIN_POOL=2
PG_FLEET_ACQUIRE_POOL=30000
PG_FLEET_IDLE_POOL=10000

# PostgreSQL - CMS Database
PG_CMS_HOST=localhost
PG_CMS_PORT=5432
PG_CMS_USER=ondelivery
PG_CMS_PASSWORD=your_strong_password
PG_CMS_DB=cms_db
PG_CMS_dialect=postgres
PG_CMS_MAX_POOL=10
PG_CMS_MIN_POOL=2
PG_CMS_ACQUIRE_POOL=30000
PG_CMS_IDLE_POOL=10000
```

```
# PostgreSQL - Mitra Database
PG_MITRA_HOST=localhost
PG_MITRA_PORT=5432
PG_MITRA_USER=ondelivery
PG_MITRA_PASSWORD=your_strong_password
PG_MITRA_DB=mitra_db
PG_MITRA_DIALECT=postgres
PG_MITRA_MAX_POOL=10

# MongoDB (if used)
MONGO_MITRA=mongodb://ondelivery:your_strong_password@localhost:27017/mitra_db

# JWT Configuration
JWT_SECRET=your_very_long_random_secret_key_min_32_characters
JWT_EXPIRATION=86400

# Firebase Configuration (if used)
FIREBASE_SERVICE_ACCOUNT_PATH=/home/ondelivery/firebase-service-account.json
FIREBASE_DATABASE_URL=https://your-project.firebaseio.com
```

Running in Production

Start Application

With PM2:

```
cd /home/ondelivery/on-internal-api
pm2 start ecosystem.config.js
pm2 save
```

With systemd:

```
sudo systemctl start on-internal-api
sudo systemctl status on-internal-api
```

Verify Application

```
# Check if application is running
curl http://localhost:3601/
# Should return: {"message":"sunshine"}

# Check logs
pm2 logs on-internal-api
# OR
sudo journalctl -u on-internal-api -f
```

Reverse Proxy Setup

Nginx Configuration

Create Nginx config:

```
sudo nano /etc/nginx/sites-available/on-internal-api
```

Configuration:

```
upstream on_internal_api {
    # If using PM2 cluster mode
    server 127.0.0.1:3601;

    # Add more if running multiple instances manually
    # server 127.0.0.1:3602;
    # server 127.0.0.1:3603;
}

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

    # Redirect to HTTPS
    return 301 https://$server_name$request_uri;
}

server {
```

```
listen 443 ssl http2;
server_name api.yourdomain.com;

# SSL Configuration
ssl_certificate /etc/letsencrypt/live/api.yourdomain.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/api.yourdomain.com/privkey.pem;
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers HIGH:!aNULL:!MD5;
ssl_prefer_server_ciphers on;

# Security Headers
add_header X-Frame-Options "SAMEORIGIN" always;
add_header X-Content-Type-Options "nosniff" always;
add_header X-XSS-Protection "1; mode=block" always;

# Client upload size
client_max_body_size 10M;

# Timeouts
proxy_connect_timeout 60s;
proxy_send_timeout 60s;
proxy_read_timeout 60s;

# Proxy settings
location / {
    proxy_pass http://on_internal_api;
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection 'upgrade';
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;
    proxy_cache_bypass $http_upgrade;
}

# WebSocket support
location /ws {
    proxy_pass http://on_internal_api;
    proxy_http_version 1.1;
```

```
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "Upgrade";
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_read_timeout 86400;
    }
}
```

Enable site:

```
sudo ln -s /etc/nginx/sites-available/on-internal-api /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl reload nginx
```

SSL/TLS Configuration

Using Let's Encrypt (Recommended)

```
# Install Certbot
sudo apt install -y certbot python3-certbot-nginx

# Obtain certificate
sudo certbot --nginx -d api.yourdomain.com

# Test auto-renewal
sudo certbot renew --dry-run
```

Manual Certificate Installation

If using a purchased certificate:

```
# Copy certificate files
sudo cp fullchain.pem /etc/ssl/certs/api.yourdomain.com.crt
sudo cp privkey.pem /etc/ssl/private/api.yourdomain.com.key
sudo chmod 644 /etc/ssl/certs/api.yourdomain.com.crt
sudo chmod 600 /etc/ssl/private/api.yourdomain.com.key
```

```
# Update Nginx config with certificate paths
sudo nano /etc/nginx/sites-available/on-internal-api
```

Monitoring

Application Monitoring with PM2

```
# Install PM2 Plus for monitoring (optional)
pm2 install pm2-logrotate

# Configure log rotation
pm2 set pm2-logrotate:max_size 10M
pm2 set pm2-logrotate:retain 7
pm2 set pm2-logrotate:compress true
```

System Monitoring

Install monitoring tools:

```
sudo apt install -y htop iotop nethogs
```

Monitor resources:

```
# CPU and memory
htop

# Disk I/O
iotop

# Network usage
nethogs
```

Log Management

Create log directory:

```
mkdir -p /home/ondelivery/on-internal-api/logs
```

Configure log rotation:

```
sudo nano /etc/logrotate.d/on-internal-api
```

```
/home/ondelivery/on-internal-api/logs/*.log {
    daily
    rotate 14
    compress
    delaycompress
    notifempty
    create 0640 ondelivery ondelivery
    sharedscripts
    postrotate
        pm2 reloadLogs
    endscript
}
```

External Monitoring (Recommended)

Consider using:

- **New Relic** - Application performance monitoring
- **Datadog** - Infrastructure and application monitoring
- **Sentry** - Error tracking
- **UptimeRobot** - Uptime monitoring

Backup Strategy

Database Backups

Automated PostgreSQL backup script:

```
#!/bin/bash
# /home/ondelivery/scripts/backup-databases.sh
```

```
BACKUP_DIR="/home/ondelivery/backups/databases"
DATE=$(date +%Y%m%d_%H%M%S)
RETENTION_DAYS=30

# Create backup directory
mkdir -p $BACKUP_DIR

# Backup each database
pg_dump -h localhost -U ondelivery sunshine_db > $BACKUP_DIR/sunshine_db_$DATE.sql
pg_dump -h localhost -U ondelivery fleet_db > $BACKUP_DIR/fleet_db_$DATE.sql
pg_dump -h localhost -U ondelivery cms_db > $BACKUP_DIR/cms_db_$DATE.sql
pg_dump -h localhost -U ondelivery mitra_db > $BACKUP_DIR/mitra_db_$DATE.sql

# Compress backups
gzip $BACKUP_DIR/*_$DATE.sql

# Delete old backups
find $BACKUP_DIR -name "*.sql.gz" -mtime +$RETENTION_DAYS -delete

echo "Database backup completed: $DATE"
```

Make executable and schedule:

```
chmod +x /home/ondelivery/scripts/backup-databases.sh

# Add to crontab (daily at 2 AM)
crontab -e
# Add: 0 2 * * * /home/ondelivery/scripts/backup-databases.sh
```

File Backups

```
#!/bin/bash
# Backup resources directory
tar -czf /home/ondelivery/backups/resources_$(date +%Y%m%d).tar.gz \
    /home/ondelivery/on-internal-api/resources

# Keep last 7 days
find /home/ondelivery/backups -name "resources_*.tar.gz" -mtime +7 -delete
```

Troubleshooting

Application Won't Start

```
# Check logs
pm2 logs on-internal-api --lines 100

# Check if port is available
sudo lsof -i :3601

# Check environment variables
pm2 env 0

# Test database connections
psql -h localhost -U ondelivery -d sunshine_db
```

High Memory Usage

```
# Check memory usage
pm2 list
free -h

# Restart application
pm2 restart on-internal-api

# If issue persists, reduce max_memory_restart in ecosystem.config.js
```

Database Connection Errors

```
# Check PostgreSQL status
sudo systemctl status postgresql

# Check connections
sudo -u postgres psql -c "SELECT count(*) FROM pg_stat_activity;"
```

```
# Check connection limits
sudo -u postgres psql -c "SHOW max_connections;"

# Increase if needed in postgresql.conf
```

SSL Certificate Issues

```
# Test certificate
sudo certbot certificates

# Renew if needed
sudo certbot renew

# Check Nginx config
sudo nginx -t
```

Rollback Procedures

Quick Rollback

```
cd /home/ondelivery/on-internal-api

# Checkout previous version
git log --oneline  # Find commit hash
git checkout <previous-commit-hash>

# Install dependencies
npm ci --only=production

# Restart application
pm2 restart on-internal-api
```

Database Rollback

```
# Restore from backup
gunzip < /home/ondelivery/backups/databases/sunshine_db_20240115_020000.sql.gz | \
    psql -h localhost -U ondelivery sunshine_db
```

Performance Tuning

Node.js Optimization

```
// ecosystem.config.js
module.exports = {
  apps: [{
    name: 'on-internal-api',
    script: './server.js',
    instances: 'max',
    exec_mode: 'cluster',
    node_args: [
      '--max-old-space-size=2048',  // Max heap size
      '--optimize-for-size',
      '--gc-interval=100'
    ]
  }]
};
```

Database Query Optimization

- Add indexes on frequently queried columns
- Use connection pooling effectively
- Implement caching for read-heavy operations
- Use database query monitoring

Caching Strategy (Redis - Optional)

```
# Install Redis
sudo apt install -y redis-server
```

```
# Configure Redis
sudo nano /etc/redis/redis.conf
# Set maxmemory and maxmemory-policy

# Restart Redis
sudo systemctl restart redis
```

Post-Deployment Verification

After deployment, verify:

- ☐ Application is running and accessible
- ☐ All API endpoints working
- ☐ Database connections established
- ☐ File uploads working
- ☐ WebSocket connections working
- ☐ SSL certificate valid
- ☐ Logs are being written
- ☐ Scheduled tasks running
- ☐ Backups configured and tested
- ☐ Monitoring alerts configured

Document Version: 1.0.0
Last Updated: 2026-02-04

For deployment support, contact the DevOps team.