Droppin Delivery Management System - Technical Documentation

Comprehensive Technical Architecture and Implementation Details

Droppin Development Team

2025

Contents

1 Executive Summary **4**
1.1 System Overview 4

2 System Architecture **5**
2.1 High-Level Architecture 5
2.2 Technology Stack Summary 5

3 Backend Architecture **6**
3.1 Core Technologies 6
3.1.1 Node.js and Express.js 6
3.1.2 Database Layer 6
3.1.3 Authentication & Security 6
3.1.4 Middleware Architecture 6
3.1.5 Database Models 7
3.1.6 API Routes Structure 7
3.1.7 Database Migrations 8
3.1.8 Scheduled Tasks 8
3.1.9 Environment Configuration 8

4 Frontend Architecture **9**
4.1 Web Frontend (React 19.1.0) 9
4.1.1 Core Technologies 9
4.1.2 Application Structure 9
4.1.3 API Integration 10

5 Mobile Frontend Architecture **11**
5.1 Mobile-Optimized React Application 11
5.1.1 Core Technologies 11
5.1.2 Mobile-Specific Features 11
5.1.3 Mobile Application Structure 12

6 Database Schema **13**
6.1 Entity Relationship Model 13
6.1.1 Core Entities 13
6.1.2 Relationships 14
6.1.3 Database Optimizations 14

7 API Documentation **15**
7.1 RESTful API Endpoints 15

7.1.1 Authentication Endpoints . . . 15
7.1.2 Package Management Endpoints . . . 15
7.1.3 Shop Management Endpoints . . . 16
7.1.4 Financial Endpoints . . . 16
7.1.5 API Integration Endpoints . . . 16
7.2 API Security . . . 16
7.2.1 Authentication Methods . . . 16
7.2.2 Rate Limiting and Security . . . 17

8 Deployment and Infrastructure 18
8.1 Deployment Architecture . . . 18
8.1.1 Backend Deployment (Vercel) . . . 18
8.1.2 Frontend Deployment . . . 18
8.1.3 Environment Configuration . . . 18
8.1.4 Database Management . . . 19

9 Security Implementation 20
9.1 Authentication Security . . . 20
9.1.1 JWT Implementation . . . 20
9.1.2 Password Security . . . 20
9.1.3 API Security . . . 20
9.1.4 Role-Based Access Control . . . 20

10 Performance Optimization 21
10.1 Database Performance . . . 21
10.1.1 SQLite Optimizations . . . 21
10.1.2 Application Performance . . . 21
10.2 Monitoring and Logging . . . 21
10.2.1 Request Logging . . . 21
10.2.2 Database Monitoring . . . 21

11 Development Workflow 22
11.1 Development Environment Setup . . . 22
11.1.1 Prerequisites . . . 22
11.1.2 Installation Process . . . 22
11.1.3 Development Scripts . . . 22
11.2 Testing Strategy . . . 22
11.2.1 Frontend Testing . . . 22
11.2.2 Backend Testing . . . 22
11.3 Code Quality . . . 23
11.3.1 Linting and Formatting . . . 23
11.3.2 Version Control . . . 23

12 Integration Capabilities 24
12.1 External Integrations . . . 24
12.1.1 Shopify Integration . . . 24
12.1.2 Payment Systems . . . 24
12.1.3 Notification Systems . . . 24
12.2 API Integration Features . . . 24
12.2.1 RESTful API Design . . . 24
12.2.2 API Documentation . . . 24

13 Future Considerations 25
13.1 Scalability Planning . . . 25
13.1.1 Database Scaling . . . 25

13.1.2 Application Scaling . . . 25
13.2 Technology Upgrades . . . 25
13.2.1 Framework Updates . . . 25
13.2.2 Feature Enhancements . . . 25

1 Executive Summary

The **Droppin Delivery Management System** is a full-stack web and mobile application built with modern technologies to provide comprehensive delivery management capabilities. The system consists of three main components: a React-based web frontend, a React-based mobile frontend, and a Node.js/Express backend API with SQLite database.

1.1 System Overview

- **Architecture:** Multi-tier architecture with separate frontend and backend applications
- **Frontend Technologies:** React 19.1.0 with modern UI libraries
- **Backend Technologies:** Node.js with Express.js framework
- **Database:** SQLite with Sequelize ORM
- **Authentication:** JWT-based authentication with role-based access control
- **Deployment:** Vercel for backend, static hosting for frontends
- **Mobile Support:** Responsive web design with mobile-optimized interface

2 System Architecture

2.1 High-Level Architecture

The Droppin system follows a **client-server architecture** with the following components:

```
  Web Frontend         Mobile Frontend         Backend API
  (React 19)           (React 19)          (Node.js/Express)

- Shop Dashboard        - Mobile UI            - REST API
- Admin Panel          - Touch Optimized        - Authentication
- Driver Portal       - i18n Support          - Business Logic




                    SQLite DB

                  - User Data
                  - Package Data
                  - Financial Data
                  - Transaction Log
```

2.2 Technology Stack Summary

Component	Technology	Version	Purpose
Frontend Framework	React	19.1.0	Web application UI
Mobile Frontend	React	19.1.0	Mobile-optimized UI
Backend Framework	Node.js + Express	5.1.0	API server
Database	SQLite	5.1.7	Data persistence
ORM	Sequelize	6.37.7	Database abstraction
Authentication	JWT	9.0.2	Token-based auth
UI Library	Ant Design	5.26.1	Component library
Styling	Bootstrap	5.3.6/5.3.7	CSS framework
HTTP Client	Axios	1.9.0/1.10.0	API communication
Charts	Chart.js	4.4.9/4.5.0	Data visualization
Animations	Framer Motion	12.23.12	UI animations
QR Code	qrcode	1.5.4	QR code generation
Icons	FontAwesome	6.7.2	Icon library
Internationalization	i18next	25.2.1	Multi-language support

3 Backend Architecture

3.1 Core Technologies

3.1 Node.js and Express.js

- **Node.js:** JavaScript runtime for server-side development
- **Express.js 5.1.0:** Web application framework providing routing, middleware, and HTTP utilities
- **Morgan:** HTTP request logger middleware for development and debugging

3.1 Database Layer

3.1.2.1 SQLite Database

- **Database Engine:** SQLite 5.1.7
- **File Location:** backend/db/dropin.sqlite
- **Performance Optimizations:**
  - WAL (Write-Ahead Logging) mode for better concurrency
  - NORMAL synchronous mode for balanced performance
  - Memory-based temporary storage
  - Optimized cache size (-20000 pages)

3.1.2.2 Sequelize ORM

- **Version:** 6.37.7
- **Purpose:** Object-Relational Mapping for database operations
- **Features:**
  - Model definitions with relationships
  - Automatic migrations
  - Query building and optimization
  - Data validation and hooks

3.1 Authentication & Security

3.1.3.1 JWT Authentication

- **Library:** jsonwebtoken 9.0.2
- **Implementation:** Bearer token-based authentication
- **Security Features:**
  - Token expiration handling
  - Role-based access control
  - API key authentication for integrations

3.1.3.2 Password Security

- **Library:** bcryptjs 3.0.2
- **Implementation:** Salt rounds (10) for password hashing
- **Features:**
  - Automatic password hashing on create/update
  - Password comparison methods
  - Secure password storage

3.1 Middleware Architecture

3.1.4.1 Authentication Middleware (auth.middleware.js)

```
// JWT token verification
exports.authenticate = async (req, res, next) => {
  // Token extraction and validation
  // User lookup and verification
  // Active status checking
}

// Role-based authorization
exports.authorize = (...roles) => {
  // Role validation for protected routes
}
```

3.1.4.2 API Key Authentication (apiKeyAuth.js)

```
// Shopify and external API integration
module.exports = async function apiKeyAuth(req, res, next) {
  // API key validation
  // Shop association
}
```

3.1 Database Models

3.1.5.1 Core Models Structure **User Model:** - Primary user entity with role-based access - Fields: name, email, password, phone, role, address, approval status - Hooks: Automatic password hashing - Roles: user, shop, driver, admin

Shop Model: - Business entity for shop owners - Fields: business info, contact details, financial tracking - Financial fields: ToCollect, TotalCollected, settled amounts - API key generation for integrations

Package Model: - Central delivery entity - Complex status enum with 20+ states - Relationships: shop, driver, pickup, items - Financial tracking: COD amounts, delivery costs

Driver Model: - Delivery personnel entity - Fields: vehicle info, working area, availability - Performance tracking: assignedToday, total deliveries

Item Model: - Package contents tracking - Fields: description, quantity, individual COD amounts - Relationship: belongs to Package

Pickup Model: - Pickup scheduling and management - Many-to-many relationship with packages - Driver assignment and status tracking

MoneyTransaction Model: - Financial transaction logging - Fields: amount, type, description, timestamps - Relationships: shop, driver associations

Notification Model: - System notifications and alerts - User-specific messaging system

3.1 API Routes Structure

3.1.6.1 Route Organization

```
/api/auth          - Authentication endpoints
/api/users         - User management
/api/shops         - Shop operations
/api/drivers       - Driver management
/api/packages      - Package lifecycle
/api/items         - Item management
/api/admin         - Administrative functions
/api/info          - System information
/api/pickups       - Pickup scheduling
```

/api - General API endpoints

3.1.6.2 Key Controllers

Package Controller (79KB, 2137 lines): - Complete package lifecycle management - Status updates and transitions - Bulk operations and API integrations - Shopify integration support

Admin Controller (71KB, 2106 lines): - System administration functions - User and shop management - Analytics and reporting - System configuration

Auth Controller (13KB, 454 lines): - User authentication and registration - Password management - Role-based access control

3.1 Database Migrations

3.1.7.1 Migration System

- **Total Migrations:** 20+ migration files
- **Purpose:** Database schema evolution and data integrity
- **Key Migrations:**
  - Package table creation and modifications
  - Financial field additions
  - Status enum updates
  - Return and exchange flow additions
  - Performance optimization indexes

3.1.7.2 Index Optimization

```sql
-- Performance indexes
CREATE INDEX IF NOT EXISTS idx_packages_status ON Packages (status);
CREATE INDEX IF NOT EXISTS idx_packages_shopId ON Packages (shopId);
CREATE INDEX IF NOT EXISTS idx_packages_driverId ON Packages (driverId);
CREATE INDEX IF NOT EXISTS idx_packages_createdAt ON Packages (createdAt);
CREATE INDEX IF NOT EXISTS idx_packages_actualDeliveryTime ON Packages (actualDeliveryTime);
```

3.1 Scheduled Tasks

3.1.8.1 Cron Jobs

- **Library:** node-cron 4.2.1
- **Daily Reset:** Driver assignment counters reset at midnight
- **Purpose:** Automated maintenance and data consistency

3.1 Environment Configuration

3.1.9.1 Environment Variables

- **JWT_SECRET:** Token signing secret
- **NODE_ENV:** Environment mode (development/production)
- **PORT:** Server port (default: 5000)
- **Database:** SQLite file path configuration

---

4 Frontend Architecture

4.1 Web Frontend (React 19.1.0)

4.1 Core Technologies

4.1.1.1 React Ecosystem

- **React:** 19.1.0 - Latest React version with concurrent features
- **React DOM:** 19.1.0 - DOM rendering
- **React Router DOM:** 6.30.0 - Client-side routing
- **React Scripts:** 5.0.1 - Build tooling and development server

4.1.1.2 UI Framework and Styling

- **Ant Design:** 5.26.1 - Enterprise-grade UI component library
- **Bootstrap:** 5.3.6 - CSS framework for responsive design
- **FontAwesome:** 6.7.2 - Icon library with React integration
- **Custom CSS:** Extensive custom styling for brand consistency

4.1.1.3 State Management and Data

- **Axios:** 1.9.0 - HTTP client for API communication
- **React Context:** Built-in state management for global state
- **Local Storage:** Client-side data persistence

4.1.1.4 Visualization and Media

- **Chart.js:** 4.4.9 - Data visualization library
- **React Chart.js 2:** 5.3.0 - React wrapper for Chart.js
- **QRCode:** 1.5.4 - QR code generation for tracking
- **Framer Motion:** 12.23.12 - Animation library

4.1.1.5 Development and Testing

- **Testing Library:** Complete testing suite
  - @testing-library/react: 16.3.0
  - @testing-library/jest-dom: 6.6.3
  - @testing-library/user-event: 13.5.0
- **Web Vitals:** 2.1.4 - Performance monitoring

4.1 Application Structure

4.1.2.1 Component Architecture

```
src/
  components/        # Reusable UI components
  pages/             # Page-level components
    Shop/            # Shop-specific pages
    Driver/          # Driver-specific pages
    Admin/           # Admin panel pages
    Auth/            # Authentication pages
  services/          # API service layer
  utils/             # Utility functions
  context/           # React Context providers
  hooks/             # Custom React hooks
  assets/            # Static assets
```

4.1.2.2 Key Components

Shop Dashboard: - Package management interface - Real-time status updates - AWB printing functionality - Financial tracking dashboard

Admin Panel: - User management system - System analytics and reporting - Configuration management - Performance monitoring

Driver Portal: - Delivery assignment interface - Status update functionality - Route optimization - Performance tracking

4.1 API Integration

4.1.3.1 Service Layer (services/api.js)

```
// Axios instance configuration
const api = axios.create({
  baseURL: 'https://api.droppin-eg.com/api',
  headers: { 'Content-Type': 'application/json' }
});

// Request interceptor for authentication
api.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Response interceptor for error handling
api.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      // Handle token expiration
      localStorage.removeItem('token');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);
```

4.1.3.2 Service Categories

- **Authentication Service:** Login, registration, password management
- **Package Service:** CRUD operations, status updates, bulk operations
- **Shop Service:** Profile management, financial tracking
- **Driver Service:** Assignment management, status updates
- **Admin Service:** System administration, analytics

5 Mobile Frontend Architecture

5.1 Mobile-Optimized React Application

5.1 Core Technologies

5.1.1.1 React Mobile Stack

- **React:** 19.1.0 - Same version as web frontend for consistency
- **React DOM:** 19.1.0 - DOM rendering
- **React Router DOM:** 6.30.1 - Mobile-optimized routing

5.1.1.2 Mobile-Specific Libraries

- **i18next:** 25.2.1 - Internationalization framework
- **React i18next:** 15.5.3 - React integration for i18n
- **HTML5 QRCode:** 2.3.8 - QR code scanning functionality
- **JSQR:** 1.4.0 - QR code detection library
- **React QR Reader:** 3.0.0-beta-1 - React QR code reader component

5.1.1.3 UI and Styling

- **Ant Design:** 5.26.1 - Mobile-optimized components
- **Bootstrap:** 5.3.7 - Responsive mobile framework
- **FontAwesome:** 6.7.2 - Mobile-friendly icons
- **Custom Mobile CSS:** Touch-optimized styling

5.1 Mobile-Specific Features

5.1.2.1 Internationalization (i18n)

```
// i18n configuration
import i18n from 'i18next';
import { initReactI18next } from 'react-i18next';

const resources = {
  en: {
    translation: {
      profile: {
        title: 'Profile',
        subtitle: 'Manage your account settings',
        language: 'Language',
        switchToArabic: 'Switch to Arabic',
        switchToEnglish: 'Switch to English'
      },
      driver: {
        profile: {
          myProfile: 'My Profile',
          personalInfo: 'Personal Information',
          edit: 'Edit',
          save: 'Save Changes'
        }
      }
    }
  },
  ar: {
    translation: {
      // Arabic translations
    }
  }
};
```

5.1.2.2 QR Code Functionality

- **QR Code Generation:** For package tracking
- **QR Code Scanning:** For driver operations
- **Camera Integration:** Mobile camera access
- **Barcode Support:** Package identification

5.1.2.3 Mobile UI Components

- **Touch-Optimized Interface:** Large touch targets
- **Swipe Gestures:** Navigation and actions
- **Mobile Navigation:** Bottom tab navigation
- **Responsive Design:** Adaptive to screen sizes

5.1 Mobile Application Structure

5.1.3.1 Component Organization

```
src/
  components/        # Mobile-specific components
  pages/             # Mobile page components
    Shop/            # Mobile shop interface
    Driver/          # Mobile driver interface
    Auth/            # Mobile authentication
  services/          # Mobile API services
  utils/             # Mobile utilities
  context/           # Mobile state management
  i18n.js            # Internationalization config
```

5.1.3.2 Mobile-Specific Pages **Mobile Shop Dashboard:** - Touch-optimized package management - Mobile AWB printing - Simplified navigation - Quick action buttons

Mobile Driver Interface: - QR code scanning for packages - Status update interface - Route navigation - Performance tracking

---

6 Database Schema

6.1 Entity Relationship Model

6.1 Core Entities

6.1.1.1 User Entity

```
CREATE TABLE Users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name VARCHAR(255) NOT NULL,
  email VARCHAR(255) UNIQUE NOT NULL,
  password VARCHAR(255) NOT NULL,
  phone VARCHAR(255) NOT NULL,
  role ENUM('user', 'shop', 'driver', 'admin') DEFAULT 'user',
  street VARCHAR(255),
  city VARCHAR(255),
  state VARCHAR(255),
  zipCode VARCHAR(255),
  country VARCHAR(255),
  isApproved BOOLEAN DEFAULT false,
  isActive BOOLEAN DEFAULT true,
  lang VARCHAR(255) DEFAULT 'en',
  createdAt DATETIME,
  updatedAt DATETIME
);
```

6.1.1.2 Shop Entity

```
CREATE TABLE Shops (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  userId INTEGER REFERENCES Users(id),
  businessName VARCHAR(255) NOT NULL,
  businessType VARCHAR(255),
  address TEXT,
  logo VARCHAR(255),
  registrationNumber VARCHAR(255),
  taxId VARCHAR(255),
  contactPersonName VARCHAR(255),
  contactPersonPhone VARCHAR(255),
  contactPersonEmail VARCHAR(255),
  isVerified BOOLEAN DEFAULT false,
  ToCollect DECIMAL(10,2) DEFAULT 0.00,
  TotalCollected DECIMAL(10,2) DEFAULT 0.00,
  settelled DECIMAL(10,2) DEFAULT 0.00,
  shippingFees DECIMAL(10,2) DEFAULT 0.00,
  shownShippingFees DECIMAL(10,2),
  apiKey VARCHAR(255) UNIQUE,
  createdAt DATETIME,
  updatedAt DATETIME
);
```

6.1.1.3 Package Entity (Simplified)

```
CREATE TABLE Packages (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  shopId INTEGER REFERENCES Shops(id),
  driverId INTEGER REFERENCES Drivers(id),
  trackingNumber VARCHAR(255) UNIQUE NOT NULL,
  type ENUM('new', 'return', 'exchange') DEFAULT 'new',
  packageDescription TEXT NOT NULL,
  weight FLOAT NOT NULL,
  dimensions VARCHAR(255),
  status ENUM(
    'awaiting_schedule', 'awaiting_pickup', 'scheduled_for_pickup',
    'pending', 'assigned', 'pickedup', 'in-transit', 'delivered',
    'cancelled', 'rejected', 'return-requested', 'return-in-transit',
    'return-completed', 'exchange-awaiting-schedule', 'exchange-in-process'
  ) DEFAULT 'awaiting_schedule',
  codAmount DECIMAL(10,2) DEFAULT 0.00,
  deliveryCost DECIMAL(10,2) DEFAULT 0.00,
  shownDeliveryCost DECIMAL(10,2),
  createdAt DATETIME,
  updatedAt DATETIME
);
```

6.1 Relationships

6.1.2.1 Primary Relationships

- **User → Shop:** One-to-One (userId foreign key)
- **User → Driver:** One-to-One (userId foreign key)
- **Shop → Package:** One-to-Many (shopId foreign key)
- **Driver → Package:** One-to-Many (driverId foreign key)
- **Package → Item:** One-to-Many (packageId foreign key)
- **Shop → Pickup:** One-to-Many (shopId foreign key)
- **Driver → Pickup:** One-to-Many (driverId foreign key)

6.1.2.2 Junction Tables

- **PickupPackages:** Many-to-Many relationship between Pickups and Packages

6.1 Database Optimizations

6.1.3.1 Indexes

```
-- Performance indexes
CREATE INDEX idx_packages_status ON Packages (status);
CREATE INDEX idx_packages_shopId ON Packages (shopId);
CREATE INDEX idx_packages_driverId ON Packages (driverId);
CREATE INDEX idx_packages_createdAt ON Packages (createdAt);
CREATE INDEX idx_packages_actualDeliveryTime ON Packages (actualDeliveryTime);
```

6.1.3.2 SQLite Optimizations

```
-- Performance PRAGMAs
PRAGMA journal_mode=WAL;         -- Write-Ahead Logging
PRAGMA synchronous=NORMAL;       -- Balanced performance
PRAGMA temp_store=MEMORY;        -- Memory-based temp storage
PRAGMA cache_size=-20000;        -- 20MB cache
ANALYZE;                         -- Update query statistics
```

7 API Documentation

7.1 RESTful API Endpoints

7.1 Authentication Endpoints

7.1.1.1 POST /api/auth/login

Purpose: User authentication **Request Body:**

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

Response:

```
{
  "token": "jwt_token_here",
  "user": {
    "id": 1,
    "name": "User Name",
    "email": "user@example.com",
    "role": "shop"
  }
}
```

7.1.1.2 POST /api/auth/register

Purpose: User registration **Request Body:**

```
{
  "name": "User Name",
  "email": "user@example.com",
  "password": "password123",
  "phone": "1234567890",
  "role": "shop"
}
```

7.1 Package Management Endpoints

7.1.2.1 GET /api/packages

Purpose: Retrieve packages with filtering **Query Parameters:** - status: Filter by package status - shopId: Filter by shop ID - driverId: Filter by driver ID - page: Pagination page number - limit: Items per page

7.1.2.2 POST /api/packages

Purpose: Create new package **Request Body:**

```
{
  "packageDescription": "Package contents",
  "weight": 2.5,
  "deliveryContactName": "Recipient Name",
  "deliveryContactPhone": "1234567890",
  "deliveryAddress": "Delivery Address",
  "codAmount": 100.00,
  "deliveryCost": 15.00
}
```

7.1.2.3 PATCH /api/packages/:id/status

Purpose: Update package status **Request Body:**

```
{
  "status": "pickedup",
  "notes": "Package picked up successfully"
}
```

7.1 Shop Management Endpoints

7.1.3.1 GET /api/shops/profile **Purpose:** Get shop profile information **Headers:** Authorization: Bearer <token>

7.1.3.2 PUT /api/shops/profile **Purpose:** Update shop profile **Request Body:**

```
{
  "businessName": "Updated Business Name",
  "contactPerson": {
    "name": "Contact Name",
    "phone": "1234567890",
    "email": "contact@business.com"
  },
  "address": "Business Address"
}
```

7.1 Financial Endpoints

7.1.4.1 GET /api/shops/money-transactions **Purpose:** Get financial transaction history **Query Parameters:** - page: Pagination page - limit: Items per page - startDate: Filter start date - endDate: Filter end date - attribute: Filter by transaction attribute - changeType: Filter by change type

7.1 API Integration Endpoints

7.1.5.1 POST /api/packages/shopify **Purpose:** Shopify integration for bulk package creation **Headers:** Authorization: Bearer <api_key> **Request Body:**

```
{
  "packages": [
    {
      "packageDescription": "Product description",
      "deliveryContactName": "Customer Name",
      "deliveryContactPhone": "1234567890",
      "deliveryAddress": "Customer Address",
      "codAmount": 50.00,
      "shopifyOrderId": "shopify_order_123"
    }
  ]
}
```

7.2 API Security

7.2 Authentication Methods

7.2.1.1 JWT Token Authentication

- **Header Format:** Authorization: Bearer <jwt_token>
- **Token Expiration:** Configurable expiration time
- **Refresh Mechanism:** Token refresh on expiration

7.2.1.2 API Key Authentication

- **Header Format:** Authorization: Bearer <api_key>

- **Use Case:** External integrations (Shopify)
- **Security:** Unique API keys per shop

7.2 Rate Limiting and Security

- **CORS:** Cross-origin resource sharing enabled
- **Request Logging:** Morgan middleware for request logging
- **Error Handling:** Comprehensive error response system
- **Input Validation:** Request body validation and sanitization

---

8 Deployment and Infrastructure

8.1 Deployment Architecture

8.1 Backend Deployment (Vercel)

8.1.1.1 Vercel Configuration (vercel.json)

```
{
  "version": 2,
  "builds": [
    {
      "src": "server.js",
      "use": "@vercel/node"
    }
  ],
  "routes": [
    {
      "src": "/(.*)",
      "dest": "/server.js"
    }
  ]
}
```

8.1.1.2 Production Environment

- **Platform:** Vercel Serverless Functions
- **Runtime:** Node.js
- **Database:** SQLite file storage
- **Domain:** api.droppin-eg.com
- **SSL:** Automatic HTTPS

8.1 Frontend Deployment

8.1.2.1 Web Frontend

- **Platform:** Static hosting (Vercel/Netlify)
- **Domain:** desktop.droppin-eg.com
- **Build Process:** React Scripts build
- **Environment Variables:** API endpoint configuration

8.1.2.2 Mobile Frontend

- **Platform:** Static hosting
- **Domain:** mobile.droppin-eg.com
- **Responsive Design:** Mobile-first approach
- **PWA Features:** Progressive Web App capabilities

8.1 Environment Configuration

8.1.3.1 Development Environment

```
# Backend
NODE_ENV=development
PORT=5000
JWT_SECRET=development_secret_key
```

```
# Frontend
REACT_APP_API_URL=http://localhost:5000/api
```

8.1.3.2 Production Environment

```
# Backend
NODE_ENV=production
JWT_SECRET=production_secret_key

# Frontend
REACT_APP_API_URL=https://api.droppin-eg.com/api
```

8.1 Database Management

8.1.4.1 SQLite Database

- **File Location:** backend/db/dropin.sqlite
- **Size:** ~1.3MB (production database)
- **Backup Strategy:** File-based backup system
- **Migration System:** Sequelize migrations

8.1.4.2 Database Download Endpoint

```
// GET /download-db
app.get('/download-db', (req, res) => {
  const dbPath = path.join(__dirname, 'db', 'dropin.sqlite');
  res.download(dbPath, 'dropin.sqlite');
});
```

9 Security Implementation

9.1 Authentication Security

9.1 JWT Implementation

- **Algorithm:** HMAC SHA256
- **Secret Management:** Environment variable configuration
- **Token Structure:** Standard JWT with user ID and role
- **Expiration:** Configurable token lifetime

9.1 Password Security

- **Hashing Algorithm:** bcrypt with salt rounds (10)
- **Implementation:** Automatic hashing on create/update
- **Password Validation:** Minimum requirements enforcement
- **Comparison Method:** Secure password comparison

9.1 API Security

9.1.3.1 CORS Configuration

```
app.use(cors({
  origin: ['https://desktop.droppin-eg.com', 'https://mobile.droppin-eg.com'],
  credentials: true
}));
```

9.1.3.2 Request Validation

- **Input Sanitization:** Request body validation
- **SQL Injection Prevention:** Sequelize ORM protection
- **XSS Protection:** Input sanitization and output encoding

9.1 Role-Based Access Control

9.1.4.1 User Roles

- **Admin:** Full system access
- **Shop:** Shop management and package operations
- **Driver:** Delivery operations and status updates
- **User:** Basic user functionality

9.1.4.2 Authorization Middleware

```
exports.authorize = (...roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ message: 'Unauthorized access' });
    }
    next();
  };
};
```

10 Performance Optimization

10.1 Database Performance

10.1 SQLite Optimizations

- **WAL Mode:** Write-Ahead Logging for better concurrency
- **Cache Configuration:** 20MB memory cache
- **Index Strategy:** Strategic indexes on frequently queried columns
- **Query Optimization:** Efficient query patterns

10.1 Application Performance

10.1.2.1 Frontend Optimizations

- **Code Splitting:** React lazy loading
- **Bundle Optimization:** Webpack optimization
- **Caching Strategy:** Browser caching for static assets
- **Image Optimization:** Compressed images and lazy loading

10.1.2.2 Backend Optimizations

- **Connection Pooling:** Database connection management
- **Response Compression:** Gzip compression
- **Caching Headers:** Appropriate cache headers
- **Async Operations:** Non-blocking I/O operations

10.2 Monitoring and Logging

10.2 Request Logging

- **Morgan Middleware:** HTTP request logging
- **Custom Logging:** Application-specific logging
- **Error Tracking:** Comprehensive error logging
- **Performance Metrics:** Response time monitoring

10.2 Database Monitoring

- **Query Logging:** Development environment query logging
- **Performance Analysis:** SQLite ANALYZE command
- **Index Usage:** Index utilization monitoring
- **Connection Monitoring:** Database connection tracking

11 Development Workflow

11.1 Development Environment Setup

11.1 Prerequisites

- **Node.js:** Version 16+ recommended
- **npm:** Package manager
- **Git:** Version control

11.1 Installation Process

```
# Backend setup
cd backend
npm install
npm run dev

# Frontend setup
cd frontend
npm install
npm start

# Mobile frontend setup
cd mobile-frontend
npm install
npm start
```

11.1 Development Scripts

11.1.3.1 Backend Scripts

```
{
  "start": "node server.js",
  "dev": "nodemon server.js",
  "migrate": "node scripts/run-migrations.js"
}
```

11.1.3.2 Frontend Scripts

```
{
  "start": "react-scripts start",
  "build": "react-scripts build",
  "test": "react-scripts test"
}
```

11.2 Testing Strategy

11.2 Frontend Testing

- **Testing Library:** React Testing Library
- **Jest:** JavaScript testing framework
- **User Event Testing:** User interaction testing
- **Component Testing:** Isolated component testing

11.2 Backend Testing

- **Unit Testing:** Individual function testing

- **Integration Testing:** API endpoint testing
- **Database Testing:** Model and migration testing

11.3 Code Quality

11.3 Linting and Formatting

- **ESLint:** JavaScript linting
- **Prettier:** Code formatting
- **Husky:** Git hooks for code quality
- **Code Review:** Peer review process

11.3 Version Control

- **Git:** Version control system
- **Branching Strategy:** Feature branch workflow
- **Commit Standards:** Conventional commit messages
- **Release Management:** Semantic versioning

12 Integration Capabilities

12.1 External Integrations

12.1 Shopify Integration

- **API Endpoint:** /api/packages/shopify
- **Authentication:** API key-based authentication
- **Bulk Operations:** Multiple package creation
- **Order Synchronization:** Shopify order ID tracking

12.1 Payment Systems

- **COD Tracking:** Cash on Delivery management
- **Settlement System:** Automated settlement processing
- **Transaction Logging:** Comprehensive financial tracking
- **Revenue Analytics:** Financial reporting and analytics

12.1 Notification Systems

- **Email Notifications:** Status update notifications
- **SMS Integration:** Mobile notifications
- **In-App Notifications:** Real-time system notifications
- **Webhook Support:** External system notifications

12.2 API Integration Features

12.2 RESTful API Design

- **Standard HTTP Methods:** GET, POST, PUT, PATCH, DELETE
- **Resource-Based URLs:** RESTful endpoint structure
- **Status Codes:** Standard HTTP status codes
- **Error Handling:** Consistent error response format

12.2 API Documentation

- **Endpoint Documentation:** Comprehensive API documentation
- **Request/Response Examples:** Detailed examples
- **Authentication Guide:** Integration authentication
- **Rate Limiting:** API usage guidelines

---

13 Future Considerations

13.1 Scalability Planning

13.1 Database Scaling

- **Migration Path:** SQLite to PostgreSQL/MySQL
- **Connection Pooling:** Advanced connection management
- **Read Replicas:** Read-only database replicas
- **Caching Layer:** Redis integration

13.1 Application Scaling

- **Microservices:** Service decomposition
- **Load Balancing:** Multiple server instances
- **CDN Integration:** Content delivery network
- **Container Deployment:** Docker containerization

13.2 Technology Upgrades

13.2 Framework Updates

- **React Updates:** Latest React features
- **Node.js Updates:** LTS version upgrades
- **Dependency Updates:** Regular dependency updates
- **Security Patches:** Timely security updates

13.2 Feature Enhancements

- **Real-time Features:** WebSocket integration
- **Advanced Analytics:** Business intelligence
- **Mobile App:** Native mobile applications
- **AI Integration:** Machine learning features

This comprehensive technical documentation provides detailed insights into the Droppin Delivery Management System's architecture, implementation, and technical specifications.

Document Version: 1.0
Last Updated: 2025
Prepared by: Droppin Development Team