# Software Developer Offboarding Documentation

## MyGooners E-Commerce Platform

**Document Version:** 1.0 **Last Updated:** January 2025 **Project:** MyGooners - E-Commerce & Services Marketplace Platform

> **? PDF Version:** To generate a PDF version of this documentation, run:

> `` `bash`

> php artisan docs:generate-pdf

> `

> This will create SOFTWARE*DEVELOPER*OFFBOARDING_DOCUMENTATION.pdf` in the project root.`

>

> You can also specify a custom input file and output filename:

> ` `bash`

> php artisan docs:generate-pdf path/to/file.md --output=custom-name.pdf

> `

---

## Executive Summary (For Non-Technical Stakeholders)

## What is MyGooners?

MyGooners is an **online marketplace platform** that combines two main features:

- **E-Commerce Shop** - Customers can browse and purchase physical products
- **Services Marketplace** - Customers can browse and request services from approved sellers

## Who Uses This System?

### Three Main User Types:

- **Customers (Regular Users)**

  - Browse products and services

  - Add items to shopping cart

  - Make purchases using credit cards or bank transfers

  - Track their orders

  - Request refunds if needed

  - Save favorite items

- **Sellers**

  - Apply to become a seller (requires approval)

  - List their services on the marketplace

  - Manage their service listings

  - Receive service requests from customers

- **Administrators**

  - Manage the entire platform

  - Approve or reject seller applications

  - Approve or reject service listings

  - Process orders and refunds

  - Manage products, articles, and content

  - View system statistics and logs

## Key Business Processes

### 1. Customer Shopping Flow: `

```
Customer browses products/services
? Adds items to cart
? Proceeds to checkout
? Enters shipping/billing information
? Chooses payment method (Stripe or ToyyibPay)
? Completes payment
? Receives order confirmation
? Tracks order status
? Receives delivery (or requests refund if needed)
```

` **2. Seller Application Flow:** `

```
User applies to become seller
? Submits business information and documents
? Admin reviews application
? Admin approves or rejects
? Approved sellers can list services
? Services go through approval process
? Approved services appear on marketplace
```

` **3. Order Management Flow:** `

```
Order created (pending payment)
? Payment received (order processing)
? Admin prepares order (processing)
? Order shipped (shipped status)
? Customer receives order (delivered)
? OR Customer requests refund (if needed)
```

`

## Main Features

- **Product Catalog** - Browse and purchase products with different variations (sizes, colors, etc.)
- **Service Listings** - Browse and request services from verified sellers
- **Shopping Cart** - Save items before checkout
- **Payment Processing** - Secure payment via Stripe (international) or ToyyibPay (Malaysia)
- **Order Tracking** - Customers can see order status in real-time
- **Refund System** - Customers can request refunds with proof
- **Content Management** - Blog articles and video gallery
- **Newsletter** - Email subscription system
- **Admin Dashboard** - Complete management interface

## Important Business Rules

- **Seller Approval Required** - Users must be approved before they can sell services
- **Service Approval Required** - All services must be approved by admin before appearing
- **Automatic Order Management:**

  - Orders pending payment for more than 24 hours are automatically cancelled

  - Orders shipped for more than 7 days are automatically marked as delivered

- **Payment Methods:**

  - Stripe: For credit card payments (international)

  - ToyyibPay: For FPX (Malaysian bank transfers) and credit cards

## System Status

? **Production Ready** - The system is live and operational

- All core features are working
- Payment processing is functional
- Order management is automated
- Admin panel is fully operational

## For Non-Technical Readers

If you're not a technical person, here's what you need to know:

- **This document contains technical details** - Some sections are for developers
- **Focus on the sections that matter to you:**

- Executive Summary (you're reading it!)

- User/Admin Manual (how to use the system)

- Operations (maintenance tasks)

- Access & Accounts (who has access to what)

- **Skip technical sections** if they don't apply to your role
- **Ask questions** - Don't hesitate to ask the technical team for clarification

---

## Table of Contents

---

# Project & System

## High-Level Architecture

## System Overview

### What is MyGooners?

MyGooners is a Laravel-based e-commerce and services marketplace platform. Think of it as a combination of:

- **An online shop** (like Amazon) where customers can buy products
- **A services marketplace** (like Fiverr) where customers can request services from sellers
- **A content platform** with blog articles and videos

### What can users do?

- Purchase products (e-commerce shop)
- Browse and request services (services marketplace)
- Read articles/blog content
- Watch video content
- Manage orders, refunds, and favorites
- Apply to become sellers (requires admin approval)

### Why two payment gateways?

- **Stripe:** For international customers and credit card payments
- **ToyyibPay:** For Malaysian customers, supports FPX (direct bank transfers) which is popular locally
- Having both gives better coverage and lower fees for the local market

## Technology Stack

**What technologies are used and why? Backend:**

- **Framework: Laravel 12.0** - A PHP framework that makes development faster and more secure. Provides built-in features like authentication, database management, and routing.
- **PHP Version: 8.0+** - The programming language. Version 8.0+ is required for modern features and better performance.
- **Database: MySQL/MariaDB (production), SQLite (development)** - Stores all data (users, products, orders, etc.). MySQL for production (live server), SQLite for local development (easier setup).
- **Package Manager: Composer** - Downloads and manages PHP libraries (like npm for JavaScript).

**Frontend:**

- **CSS Framework: Tailwind CSS 4.0** - Makes styling easier with utility classes. Instead of writing custom CSS, you use pre-built classes.
- **Build Tool: Vite 6.2.4** - Compiles and optimizes CSS/JavaScript files for production. Makes the website load faster.
- **JavaScript: Vanilla JS with Axios** - Vanilla JS means plain JavaScript (no framework). Axios is used for making API calls (like fetching data).

**Key Dependencies:**

- stripe/stripe-php `(^17.4) - Payment processing`
- barryvdh/laravel-dompdf `(^3.1) - PDF generation for invoices`
- laravel/socialite `(^5.12) - Google OAuth integration`
- doctrine/dbal `(^4.3) - Database abstraction`

## System Architecture Diagram

**Client Layer (Frontend)** `

+----------+ +----------+ +----------+ +----------+

| Shop | | Services | | Blog | | Videos |

+----------+ +----------+ +----------+ +----------+

|

v

` **Application Layer (Laravel)** `

+-------------+ +-------------+ +-------------+

| Controllers | | Services | | Models |

| (34 files) | | (4 files) | | (19 files) |

+-------------+ +-------------+ +-------------+

+-------------+ +-------------+ +-------------+

| Middleware | | Mail | | Notifications |

+-------------+ +-------------+ +-------------+

|

v

` **Integration Layer** `

+-------------+ +-------------+ +-------------+

| Stripe | | ToyyibPay | | Google OAuth |

| Payment | | Payment | | |

+-------------+ +-------------+ +-------------+

|

v

` **Data Layer** `

+-------------+ +-------------+ +-------------+

| MySQL | | Storage | | Cache |

| Database | | (Files) | | |

+-------------+ +-------------+ +-------------+

`

## Core Modules

- **E-Commerce Module**

  - Product catalog with variations

  - Shopping cart

  - Checkout (regular & direct)

  - Order management

  - Invoice generation

- **Services Marketplace Module**

  - Service listings

  - Seller application system

  - Service approval workflow

  - Service reviews

- **Content Management Module**

  - Article/blog system

  - Video gallery

  - Rich text editor with social media embeds

  - Newsletter system

- **User Management Module**

  - User authentication (email/password + Google OAuth)

  - Seller application process

  - Admin panel

  - User profiles

- **Payment Module**

  - Stripe integration

  - ToyyibPay integration

  - Payment retry mechanism

  - Webhook handling

- **Order Management Module**

  - Order lifecycle management

  - Auto-cancellation (24 hours)

  - Auto-delivery marking (7 days)

```
- Refund system
```

- **Admin Module**

  ```
  - Dashboard with statistics

  - Content management

  - Order management

  - User management

  - Settings management

  - Log viewer

  ---
  ```

## Project Setup Guide

## Prerequisites

**Required Software:**

- PHP 8.0 or higher
- Composer 2.0 or higher
- Node.js 16+ and npm
- MySQL 5.7+ or MariaDB
- Git

**Required PHP Extensions:**

- BCMath
- Ctype
- JSON
- Mbstring
- OpenSSL
- PDO
- Tokenizer
- XML
- cURL
- GD
- ZIP
- Fileinfo

## Installation Steps

- **Clone the Repository**

```bash
git clone
cd mygooners
```

- **Install PHP Dependencies**

```bash
composer install
```

- **Install Node Dependencies**

```bash
npm install
```

- **Environment Configuration**

```bash
cp .env.example .env
php artisan key:generate
```

- **Configure .env File**

  See Environment Configuration section for details.

- **Database Setup**

```bash
php artisan migrate
php artisan db:seed
```

- **Storage Link**

```bash
php artisan storage:link
```

- **Build Frontend Assets**

` bash

```
npm run build

# or for development:

npm run dev
```

`

- **Start Development Server**

  ` bash

  ```
  php artisan serve
  ```

  `

## Environment Variables (.env)

**Application Settings:** `env

APP_NAME=MyGooners

APP_ENV=local

APP_KEY=base64:...

APP_DEBUG=true

APP_URL=http://localhost:8000

APP*TIMEZONE=Asia/Kuala*Lumpur

APP_LOCALE=ms

APP*FALLBACK*LOCALE=en

` **Database Configuration:** `env

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=mygooners

DB_USERNAME=root

DB_PASSWORD=

` **Payment Gateways:** `env

## Stripe

STRIPE*KEY=pk*test_...

STRIPE*SECRET=sk*test_...

STRIPE*WEBHOOK*SECRET=whsec_...

# ToyyibPay

TOYYIBPAY*SECRET*KEY=your*secret*key

TOYYIBPAY*CATEGORY*CODE=your*category*code

TOYYIBPAY*BASE*URL=https://toyyibpay.com

` **Email Configuration:** ` env

MAIL_MAILER=smtp

MAIL_HOST=mail.mygooners.my

MAIL_PORT=587

MAIL_USERNAME=noreply@mygooners.my

MAIL*PASSWORD=your*password

MAIL_ENCRYPTION=tls

MAIL*FROM*ADDRESS=noreply@mygooners.my

MAIL*FROM*NAME="${APP_NAME}"

` **Google OAuth:** ` env

GOOGLE*CLIENT*ID=your*client*id

GOOGLE*CLIENT*SECRET=your*client*secret

GOOGLE*REDIRECT*URI=http://localhost:8000/auth/google/callback

` **Session Configuration:** ` env

SESSION_DRIVER=file

SESSION_LIFETIME=480

`

---

## Getting Started Guide (For New Developers)

## Welcome, New Developer! ?

This guide will help you get up and running quickly. Follow these steps in order.

## Prerequisites Checklist

Before you start, make sure you have:

- [ ] **PHP 8.0+** installed (check with `php -v`)
- [ ] **Composer** installed (check with `composer --version`)
- [ ] **Node.js 16+** and npm installed (check with `node -v` and `npm -v`)
- [ ] **MySQL** or **SQLite** database available
- [ ] **Git** installed (check with `git --version`)
- [ ] A code editor (VS Code, PhpStorm, etc.)

**Don't have these?** See Project Setup Guide for installation instructions.

### First Day Checklist

**Morning (2-3 hours):**

- [ ] Read the Executive Summary to understand what the system does
- [ ] Set up your local development environment (follow steps below)
- [ ] Run the application and see it working
- [ ] Create a test account and explore the frontend

**Afternoon (2-3 hours):**

- [ ] Explore the admin panel
- [ ] Create a test product and order
- [ ] Review the code structure (see Folder Structure)
- [ ] Read about Critical Code Areas

## Step-by-Step Setup

Step 1: Clone and Install (15 minutes)

` bash

## Clone the repository

```
git clone

cd mygooners
```

## Install PHP dependencies

```
composer install
```

## Install Node dependencies

```
npm install
```

` **What's happening?**

- composer install `downloads all PHP libraries the project needs`
- npm install `downloads all JavaScript/CSS tools needed for the frontend`

### Step 2: Environment Setup (10 minutes)

` `bash`

## Copy the example environment file

```
cp .env.example .env
```

## Generate application key

```
php artisan key:generate
```

` **What's happening?**

- .env file stores your local configuration (database, API keys, etc.)
- The application key encrypts sensitive data

**Next:** Edit .env file and set:

- DB_DATABASE - Your database name
- DB_USERNAME - Your database username
- DB_PASSWORD - Your database password

## Step 3: Database Setup (5 minutes)

`` ` bash``

## Create database tables

```
php artisan migrate
```

## Add sample data (optional but helpful)

```
php artisan db:seed
```

` **What's happening?**

- migrate `creates all the database tables`
- db:seed `adds sample products, users, etc. for testing`

### Step 4: Storage Link (1 minute)

` bash

## Create symbolic link for file storage

```
php artisan storage:link
```

` **What's happening?** This makes uploaded files (images, documents) accessible via the web.

Step 5: Build Frontend (2 minutes)

`bash

## Build CSS and JavaScript files

```
npm run build
```

## OR for development (auto-rebuilds on changes):

```
npm run dev
```

`

Step 6: Start the Server (1 minute)

` bash

## Start Laravel development server

php artisan serve

` **Success!** Open http://localhost:8000 in your browser. You should see the MyGooners homepage.

## Learning Path (First Week)

**Day 1: Understanding the System**

- ? Complete setup (you just did this!)
- Read System Overview
- Explore the frontend as a user
- Explore the admin panel

**Day 2: Code Structure**

- Review Folder Structure
- Understand MVC pattern (Models, Views, Controllers)
- Read about Core Modules
- Look at a simple controller (e.g., `HomeController` )

**Day 3: Database & Models**

- Review Database Schema
- Understand Eloquent models
- Look at a model file (e.g., `User.php` , `Product.php` )
- Understand relationships (hasMany, belongsTo)

**Day 4: Routes & Controllers**

- Review `routes/web.php`
- Understand how routes work
- Look at a controller method
- Trace a request: Route ? Controller ? View

**Day 5: Payment Flow**

- Read Payment Processing
- Understand Stripe integration
- Understand ToyyibPay integration
- Test a payment (use test mode!)

## Common First-Time Issues

**Issue: "Class not found" errors**

- **Solution:** Run

composer dump-autoload **Issue: Database connection error**

- **Solution:** Check your

.env file has correct database credentials **Issue: "Storage link already exists"**

- **Solution:** Delete

public/storage and run php artisan storage:link again **Issue: CSS/JS not loading**

- **Solution:** Run

npm run build or npm run dev **Issue: "Permission denied" errors**

- **Solution:** Make sure

storage/ and bootstrap/cache/ folders are writable

## Understanding Laravel Basics

**If you're new to Laravel, here are key concepts:**

- **MVC Pattern:**

  - **Model** = Database tables (e.g.,

  `User`, `Product`, `Order`)

  - **View** = HTML templates (in

  `resources/views/`)

  - **Controller** = Business logic (in

  `app/Http/Controllers/`)
- **Routes:**

  - Defined in

  `routes/web.php`

  - Map URLs to controller methods

  - Example:

  `/shop` ? `ProductController@index`
- **Eloquent ORM:**

  - Laravel's way of talking to the database

  - Example:

  `User::find(1)` gets user with ID 1

  - Relationships:

  `$user->orders` gets all orders for a user
- **Blade Templates:**

  - Laravel's templating engine

  - Files in

  `resources/views/`

  - Mix PHP and HTML:

  `{{ $variable }}` outputs data

## Where to Get Help

- **This Documentation** - Start here!
- **Laravel Documentation** - https://laravel.com/docs
- **Stack Overflow** - For specific error messages
- **Team Members** - Don't hesitate to ask!

## Next Steps

Once you're comfortable with the basics:

- Read Critical Code Areas to understand important parts
- Review Known Bugs & Limitations
- Check out the API Documentation
- Explore the Services folder to see business logic

---

## API Documentation

## Authentication

The application uses Laravel's built-in authentication system with session-based authentication.

**User Roles:**

- user - Regular customer
- admin - Administrator
- super_admin - Super administrator

## Key Endpoints

`Public Endpoints`

**Home & Content:**

- GET / `- Home page`
- GET /blog `- Blog listing`
- GET /blog/{slug} `- Article detail`
- GET /shop `- Product shop`
- GET /shop/{slug} `- Product detail`
- GET /services `- Services listing`
- GET /services/{slug} `- Service detail`
- GET /videos `- Video gallery` **Authentication:**
- GET /login `- Login page`
- POST /login `- Login`
- GET /register `- Registration page`
- POST /register `- Register`
- GET /auth/google `- Google OAuth redirect`
- GET /auth/google/callback `- Google OAuth callback`

Authenticated Endpoints

**Cart:**

- GET /cart `- View cart`
- POST /cart/add `- Add to cart`
- PUT /cart/update/{item} `- Update cart item`
- DELETE /cart/remove/{item} `- Remove from cart` **Checkout:**
- GET /checkout `- Checkout page`
- POST /checkout `- Create order`
- GET /checkout/orders `- User orders list`
- GET /checkout/orders/{orderId} `- Order details`
- POST /checkout/orders/{order}/cancel `- Cancel order`
- POST /checkout/orders/{order}/mark-delivered `- Mark as delivered`
- POST /checkout/orders/{order}/retry-payment `- Retry payment` **Favorites:**
- GET /favourites `- User favorites`
- POST /favourites/add `- Add favorite`
- DELETE /favourites/remove `- Remove favorite` **Refunds:**
- GET /checkout/refunds `- User refunds list`
- GET /checkout/orders/{order}/refund `- Create refund request`
- POST /checkout/orders/{order}/refund `- Submit refund request`

`Admin Endpoints`

**Dashboard:**

- GET /admin/dashboard `- Admin dashboard`
- GET /admin/dashboard/stats `- Dashboard statistics (AJAX)` **Products:**
- GET /admin/products `- Products list`
- POST /admin/products `- Create product`
- PUT /admin/products/{id} `- Update product`
- DELETE /admin/products/{id} `- Delete product`
- POST /admin/products/{id}/toggle-status `- Toggle status`
- POST /admin/products/{id}/toggle-featured `- Toggle featured` **Orders:**
- GET /admin/orders `- Orders list`
- GET /admin/orders/{id} `- Order details`
- PATCH /admin/orders/{id}/status `- Update order status`
- PATCH /admin/orders/{id}/payment-status `- Update payment status`
- GET /admin/orders/{id}/invoice `- View invoice`
- GET /admin/orders/{id}/invoice/download `- Download invoice` **Services:**
- GET /admin/services `- Services list`
- GET /admin/services/pending `- Pending services`
- POST /admin/services/{id}/approve `- Approve service`
- POST /admin/services/{id}/reject `- Reject service` **Users:**
- GET /admin/users `- Users list`
- POST /admin/users/{id}/verify `- Verify user`
- POST /admin/users/{id}/suspend `- Suspend user` **Settings:**
- GET /admin/settings `- Settings page`
- POST /admin/settings `- Update settings`

## Payment Flow

## Stripe Payment Flow

- User selects Stripe payment method
- System creates Stripe PaymentIntent
- User redirected to Stripe checkout
- Stripe webhook handles payment confirmation
- Order status updated to "paid"

### ToyyibPay Payment Flow

- User selects ToyyibPay payment method
- System creates ToyyibPay bill
- User redirected to ToyyibPay payment page
- ToyyibPay callback updates order status
- User redirected back to success page

## Error Codes

**HTTP Status Codes:**

- 200 `- Success`
- 302 `- Redirect`
- 404 `- Not Found`
- 403 `- Forbidden`
- 422 `- Validation Error`
- 500 `- Server Error` **Custom Error Messages:**

- Payment session expired: "Sesi pembayaran tidak sah atau telah tamat"
- Payment failed: "Pembayaran gagal. Sila cuba lagi."
- Order not found: "Pesanan tidak dijumpai"

---

## Coding Standards

## Naming Conventions

**Files & Directories:**

- Controllers: PascalCase (e.g., CheckoutController.php )
- Models: PascalCase (e.g., Order.php )
- Services: PascalCase (e.g., StripeService.php )
- Views: kebab-case (e.g., checkout/index.blade.php )
- Migrations: snake*case with timestamp (e.g., 20250101*create*orders*table.php ) **Code:**
- Classes: PascalCase
- Methods: camelCase
- Variables: camelCase
- Constants: UPPER*SNAKE*CASE
- Database tables: snake_case, plural
- Database columns: snake_case

## Folder Structure

`

```
mygooners/
+-- app/
| +-- Console/
| | +-- Commands/ # Artisan commands
| +-- Http/
| | +-- Controllers/
| | | +-- Admin/ # Admin controllers
| | | +-- Client/ # Client controllers
| | +-- Middleware/ # Custom middleware
| +-- Mail/ # Mail classes
| +-- Models/ # Eloquent models
| +-- Notifications/ # Notification classes
| +-- Providers/ # Service providers
| +-- Services/ # Business logic services
+-- config/ # Configuration files
+-- database/
| +-- factories/ # Model factories
| +-- migrations/ # Database migrations
| +-- seeders/ # Database seeders
+-- public/ # Public assets
+-- resources/
| +-- css/ # CSS files
| +-- js/ # JavaScript files
| +-- lang/ # Language files
| +-- views/ # Blade templates
| +-- admin/ # Admin views
| +-- client/ # Client views
| +-- layouts/ # Layout templates
+-- routes/
```

```
| +-- web.php # Web routes

| +-- console.php # Console routes (scheduled tasks)

+-- storage/ # Storage (logs, cache, files)

+-- tests/ # Tests

 `
```

### Linting Rules

**PHP:**

- Use Laravel Pint for code formatting
- Follow PSR-12 coding standards
- Maximum line length: 120 characters

**JavaScript:**

- Use ESLint (if configured)
- Follow modern JavaScript standards

### Code Organization Principles

- **Controllers:** Handle HTTP requests, delegate to services
- **Services:** Contain business logic
- **Models:** Handle data access and relationships
- **Middleware:** Handle cross-cutting concerns (auth, admin)
- **Views:** Presentation layer only

---

## Code-Level

## Work-in-Progress (WIP) Report

## Completed Features ?

- **E-Commerce System**

  - Product catalog with variations

  - Shopping cart

  - Checkout system (regular & direct)

  - Order management

  - Invoice generation and email

- **Services Marketplace**

  - Service listings

  - Seller application system

  - Service approval workflow

  - Service reviews

- **Payment Integration**

  - Stripe payment gateway

  - ToyyibPay payment gateway

  - Payment retry mechanism

  - Webhook handling

- **Order Automation**

  - Auto-cancel pending orders (24 hours)

  - Auto-mark delivered (7 days after shipping)

- **Refund System**

  - Refund request workflow

  - Admin refund management

  - Refund status tracking

- **Content Management**

  - Article/blog system

  - Rich text editor

  - Social media embeds

  - Video gallery

  - Newsletter system

- **Admin Panel**

```
- Dashboard with statistics
```

- Order management

- Product management

- Service management

- User management

- Settings management

- Log viewer

- **User Features**

- Authentication (email/password + Google OAuth)

- Favorites system

- Order tracking

- Profile management

## In-Progress Tasks ?

- **Testing**

  - Unit tests for services

  - Feature tests for controllers

  - Integration tests for payment flows

- **Performance Optimization**

  - Query optimization

  - Caching implementation

  - Image optimization

## Pending Features ?

- **Email Notifications**

  - Order status update emails

  - Refund status emails

  - Auto-cancellation notifications

- **Analytics**

  - Sales reports

  - User analytics

  - Product performance metrics

- **Mobile App**

  - API development

  - Mobile app integration

- **Advanced Features**

  - Bulk order operations

  - Shipping label generation

  - Real-time order updates (WebSocket)

  - Advanced search and filters

---

## Critical Code Areas

### 1. Payment Processing

**Location:** `app/Services/StripeService.php` , `app/Services/ToyyibPayService.php`

**Key Logic:**

- Payment intent/bill creation
- Payment verification
- Webhook handling
- Payment retry mechanism

**Important Notes:**

- Payment data is backed up in cache for 1 hour
- Session data is used as primary source, cache as fallback
- Payment IDs are only updated after successful payment creation
- Webhooks must be configured in Stripe dashboard

## 2. Order Lifecycle Management

**Location:** app/Http/Controllers/Client/CheckoutController.php **Order Statuses:**

- pending - Order created, payment pending
- processing - Payment received, order being prepared
- shipped - Order shipped
- delivered - Order delivered
- cancelled - Order cancelled
- refunded - Order refunded **Payment Statuses:**
- pending - Payment initiated
- paid - Payment successful
- failed - Payment failed **Key Methods:**
- store() - Create order
- toyyibpayReturn() - Handle ToyyibPay return
- stripeReturn() - Handle Stripe return
- retryPayment() - Retry failed payment
- cancelOrder() - Cancel order

## 3. Scheduled Tasks (Cron Jobs)

**Location:** routes/console.php , app/Console/Commands/ **Scheduled Commands:**

- **Auto-Cancel Pending Orders**

  - Command:

  orders:auto-cancel-pending

  - Schedule: Hourly

  - Logic: Cancels orders pending > 24 hours without payment

  - File:

  app/Console/Commands/AutoCancelPendingOrders.php
- **Auto-Mark Delivered**

  - Command:

  orders:auto-mark-delivered

  - Schedule: Daily at 09:00

  - Logic: Marks shipped orders as delivered after 7 days

  - File:

  app/Console/Commands/AutoMarkOrdersAsDelivered.php **Setup for Production:** `
  bash

## Add to crontab

```
0     cd /path/to/project && php artisan orders:auto-cancel-pending >> /dev/null 2>&1

0 9   * cd /path/to/project && php artisan orders:auto-mark-delivered >> /dev/null 2>&1
```

`

### 4. Invoice Generation

**Location:** `app/Services/InvoiceService.php` **Key Features:**

- PDF generation using DomPDF
- Email attachment
- Temporary file storage
- Automatic cleanup

**Important Notes:**

- Invoices are generated on-the-fly
- Files are stored temporarily in `storage/app/temp/invoices/`
- Automatic cleanup after email sending
- Requires storage link: `php artisan storage:link`

## 5. Seller Application System

**Location:** app/Http/Controllers/Admin/SellerRequestController.php **Workflow:**

- User submits seller application
- Admin reviews application
- Admin approves/rejects
- User notified of decision
- Approved users can create services

**Key Fields:**

- Business information
- ID documents
- Selfie with ID
- Business registration (for companies)

## 6. Service Approval Workflow

**Location:** app/Http/Controllers/Admin/ServiceController.php **Service Statuses:**

- pending - Awaiting approval
- active - Approved and active
- inactive - Approved but inactive
- rejected - Rejected by admin **Update Request System:**
- Sellers can request service updates
- Updates create new pending records
- Original service remains active until update approved

## 7. Refund System

**Location:** app/Http/Controllers/Client/RefundController.php , app/Http/Controllers/Admin/RefundController.php **Refund Statuses:**

- pending - Refund requested
- approved - Refund approved
- processing - Refund being processed
- completed - Refund completed
- rejected - Refund rejected **Key Features:**
- Image upload for refund proof
- Admin approval workflow
- Bank account information collection
- Status tracking

## 8. Integration Points

**Stripe Integration:**

- API Key:

config/services.php

- Webhook Secret: Required for webhook verification
- Webhook URL:

/checkout/stripe/webhook

- Events handled:

payment*intent.succeeded* , *payment*intent.payment_failed **ToyyibPay Integration:**

- Secret Key:

config/services.php

- Category Code: Required
- Callback URL:

/checkout/toyyibpay/callback

- Return URL:

/checkout/toyyibpay/return **Google OAuth:**

- Client ID & Secret:

config/services.php

- Redirect URI:

/auth/google/callback

- Scopes: email, profile

---

## Known Bugs & Limitations

## Known Issues

- **Session Timeout for Payments**

  - **Issue:** Payment session can expire during payment processing

  - **Workaround:** Extended session lifetime to 480 minutes, added cache fallback

  - **Status:** Partially resolved, monitoring needed

  - **Reference:**

  STRIPE*PAYMENT*FIXES.md
- **ToyyibPay Bill Cancellation**

  - **Issue:** ToyyibPay doesn't provide direct cancel API

  - **Workaround:** Manual intervention may be required

  - **Status:** Known limitation

  - **Reference:**

  app/Services/ToyyibPayService.php `line 307`
- **Article Content Image Paths**

  - **Issue:** Multiple possible storage locations for article images

  - **Workaround:** System checks multiple paths

  - **Status:** Working but could be optimized

  - **Reference:**

  routes/web.php `lines 632-656`
- **Storage Link Issues on cPanel**

  - **Issue:** Storage link may break after deployment

  - **Workaround:** Recreate link:

  php artisan storage:link

  - **Status:** Known deployment issue

  - **Reference:**

  CPANEL*DEPLOYMENT*GUIDE.md

## Limitations

- **Email Queue Not Implemented**

  - Emails are sent synchronously

  - May cause delays for high-volume operations

  - **Future Enhancement:** Implement queue system

- **No Real-time Updates**

  - Order status updates require page refresh

  - **Future Enhancement:** WebSocket integration

- **Limited Payment Methods**

  - Currently supports Stripe and ToyyibPay only

  - **Future Enhancement:** Add more payment gateways

- **No Bulk Operations**

  - Admin must process orders one by one

  - **Future Enhancement:** Bulk order operations

- **Image Optimization**

  - Images are not automatically optimized

  - **Future Enhancement:** Image compression and optimization

- **No API Rate Limiting**

  - API endpoints don't have rate limiting

  - **Future Enhancement:** Implement rate limiting

---

## Troubleshooting Guide

## Common Issues and Solutions

### Payment Issues

**Problem: Payment stuck in "pending" status**

- **Check:** Payment gateway logs in `storage/logs/laravel.log`
- **Check:** Verify API keys in `.env` file are correct
- **Check:** Webhook configuration in Stripe/ToyyibPay dashboard
- **Solution:** See Payment Processing section

**Problem: "Sesi pembayaran tidak sah atau telah tamat" (Invalid payment session)**

- **Cause:** Session expired during payment
- **Solution:** Session lifetime extended to 480 minutes, cache fallback implemented
- **Reference:** See STRIPE*PAYMENT*FIXES.md

**Problem: Payment webhook not working**

- **Check:** Webhook URL is correct: `https://yourdomain.com/checkout/stripe/webhook`
- **Check:** Webhook secret in `.env` matches Stripe dashboard
- **Check:** Server can receive POST requests from Stripe
- **Solution:** Test webhook in Stripe dashboard ? Developers ? Webhooks

### Database Issues

**Problem: "Table doesn't exist" error**

- **Solution:** Run `php artisan migrate`
- **Check:** Database connection in `.env` is correct
- **Check:** Database user has CREATE TABLE permissions

**Problem: Migration fails**

- **Check:** Database connection
- **Check:** Table doesn't already exist
- **Check:** Foreign key constraints
- **Solution:** Review migration file for errors

### File Upload Issues

**Problem: Images not displaying**

- **Check:** Storage link exists:

  php artisan storage:link

- **Check:** File permissions:

  chmod -R 775 storage

- **Check:** Files are in

  storage/app/public/ directory

- **Solution:** Verify

  public/storage is a symlink to storage/app/public **Problem: "Permission denied" when uploading**

- **Solution:** Set correct permissions:

  ` bash

  chmod -R 775 storage

  chmod -R 775 bootstrap/cache

  `

```
Application Errors
```

**Problem: "Class not found" errors**

- **Solution:** Run

  `composer dump-autoload`

- **Check:** Class name matches file name
- **Check:** Namespace is correct

**Problem: "Route not found"**

- **Check:** Routes are defined in

  `routes/web.php`

- **Solution:** Run

  `php artisan route:clear && php artisan route:cache` **Problem: CSS/JavaScript not loading**

- **Solution:** Run

  `npm run build` or `npm run dev`

- **Check:** Files exist in

  `public/build/` directory

- **Check:** Vite is running if using

  `npm run dev`

### Email Issues

**Problem: Emails not sending**

- **Check:** SMTP settings in `.env` file
- **Check:** Mail credentials are correct
- **Test:** Use test route `/test-email` (if available)
- **Check:** Mail logs in `storage/logs/laravel.log`

**Problem: Email timeout errors**

- **Solution:** Increase `MAIL_TIMEOUT` in `.env`
- **Check:** SMTP server is reachable
- **Check:** Firewall allows SMTP connections

### Cron Job Issues

**Problem: Scheduled tasks not running**

- **Check:** Cron job is set up in cPanel/server
- **Check:** Command path is correct
- **Check:** PHP path in cron job is correct
- **Solution:** Test command manually:

`php artisan orders:auto-cancel-pending`

**Problem: "Command not found" in cron**

- **Solution:** Use full path to PHP:

`/usr/bin/php` (check with `which php`)

- **Solution:** Use full path to artisan:

`/path/to/project/artisan`

### Performance Issues

**Problem: Slow page loads**

- **Check:** Enable caching:

```
php artisan config:cache && php artisan route:cache
```

- **Check:** Database queries (use Laravel Debugbar)
- **Check:** Image sizes (optimize large images)
- **Solution:** Enable OPcache if available

**Problem: High memory usage**

- **Check:** Image processing (resize images before upload)
- **Check:** Database queries (avoid N+1 queries)
- **Solution:** Increase PHP memory limit if needed

## Debugging Tips

- **Check Logs First**

  - Laravel logs:

  storage/logs/laravel.log

  - DomPDF logs:

  storage/logs/dompdf.log

  - Server logs: Check cPanel error logs

- **Enable Debug Mode (Development Only)**

  ```bash
  # In .env file
  APP_DEBUG=true
  ```

- **Clear All Caches**

  ```bash
  php artisan config:clear
  php artisan route:clear
  php artisan view:clear
  php artisan cache:clear
  ```

- **Test Database Connection**

  ```bash
  php artisan tinker
  DB::connection()->getPdo();
  ```

- **Check Environment Variables**

  ```bash
  php artisan config:show
  ```

## Getting Help

If you can't solve an issue:

- **Check this documentation** - Search for the error message
- **Check Laravel logs** - Look for error details
- **Search Stack Overflow** - Someone may have had the same issue
- **Ask the team** - Don't hesitate to reach out
- **Check GitHub issues** - If using open-source packages

### Temporary Workarounds

- **Composer Update Route (REMOVE IN PRODUCTION)**

  - **Location:**

  routes/web.php `lines 143-160`

  - **Purpose:** Temporary route for running composer update

  - **Action Required:** Remove before production deployment
- **Test Email Routes**

  - **Location:**

  routes/web.php `lines 783-926`

  - **Purpose:** Testing email configuration

  - **Action Required:** Consider removing or protecting with auth

  ---

# DevOps & Infrastructure

## Deployment Process

## cPanel Deployment

**Reference:** CPANEL*DEPLOYMENT*GUIDE.md **Steps:**

- **Upload Files**

  - Upload all files to

  public_html directory

  - Ensure hidden files (

  .env ) are uploaded
- **Set Permissions**

  ` bash

  chmod -R 755 public_html

  chmod -R 775 public_html/storage

  chmod -R 775 public_html/bootstrap/cache

  `
- **Install Dependencies**

  ` bash

  cd public_html

  composer install --optimize-autoloader --no-dev

  `
- **Environment Setup**

  - Create/update

  .env file

  - Set

  APP_ENV=production

  - Set

  APP_DEBUG=false

  - Configure database and mail settings
- **Database Migration**

  ` bash

  php artisan migrate --force

  php artisan db:seed --force

`

- **Optimize Application**

  ` bash

```
php artisan config:cache

php artisan route:cache

php artisan view:cache

php artisan storage:link
```

  `

- **Build Frontend**

  ` bash

```
npm run build
```

  `

## CI/CD Flow

**Current Status:** Manual deployment **Recommended CI/CD Pipeline:**

- Code push to repository
- Automated tests
- Build assets
- Deploy to staging
- Manual approval
- Deploy to production

### Rollback Steps

- **Database Rollback**

  ` bash

  php artisan migrate:rollback --step=1

  `

- **Code Rollback**

  - Revert to previous Git commit

  - Re-run deployment steps

- **Cache Clear**

  ` bash

  php artisan config:clear

  php artisan route:clear

  php artisan view:clear

  php artisan cache:clear

  `

## Common Deployment Issues

- **Storage Link Broken**

  - **Solution:**

```
php artisan storage:link
```

- **Permission Errors**

  - **Solution:** Check file permissions, ensure storage is writable

- **Config Cache Issues**

  - **Solution:**

```
php artisan config:clear && php artisan config:cache
```

- **Route Cache Issues**

  - **Solution:**

```
php artisan route:clear && php artisan route:cache
```

---

## Server/Cloud Documentation

## Server Information

**Production Server:**

- **Type:** cPanel hosting
- **Domain:** mygooners.my
- **PHP Version:** 8.1+
- **Database:** MySQL/MariaDB

### Access Locations

**File Access:**

- **FTP/SFTP:** Use cPanel file manager or FTP client
- **SSH:** If available, use SSH for command-line access

**Database Access:**

- **cPanel phpMyAdmin:** Access via cPanel
- **Direct Connection:** Use database credentials from

`.env`

## Monitoring

**Log Files:**

- **Laravel Logs:**

  storage/logs/laravel.log
- **DomPDF Logs:**

  storage/logs/dompdf.log
- **Server Logs:** Check cPanel error logs

**Monitoring Commands:** ` bash

## View Laravel logs

```
tail -f storage/logs/laravel.log
```

## Check disk space

```
df -h
```

## Check PHP processes

```
ps aux | grep php
```

`

### Backups

**Database Backups:**

- Configure via cPanel backup system
- Recommended: Daily automated backups
- Retention: 30 days minimum

**File Backups:**

- Backup `storage/app` directory
- Backup `.env` file (securely)
- Backup uploaded files

**Backup Locations:**

- Store backups off-server
- Use cloud storage (AWS S3, Google Cloud, etc.)

---

## Environment Configuration

## Environment Differences

**Development (local):** `env

```
APP_ENV=local

APP_DEBUG=true

APP_URL=http://localhost:8000

DB_CONNECTION=sqlite

SESSION_DRIVER=file
```

` **Staging:** `env

```
APP_ENV=staging

APP_DEBUG=false

APP_URL=https://staging.mygooners.my

DB_CONNECTION=mysql

SESSION_DRIVER=database
```

` **Production:** `env

```
APP_ENV=production

APP_DEBUG=false

APP_URL=https://mygooners.my

DB_CONNECTION=mysql

SESSION_DRIVER=file

SESSION_LIFETIME=480
```

`

```
```

## Configuration Notes

- **Session Configuration**

  - Production: File driver, 480 minutes lifetime

  - Development: File driver, 120 minutes lifetime
- **Cache Configuration**

  - Production: Use Redis if available

  - Development: File cache
- **Mail Configuration**

  - Production: SMTP with cPanel mail server

  - Development: Log driver for testing
- **Payment Gateways**

  - Production: Live API keys

  - Development: Test API keys
- **Error Reporting**

  - Production:

```
APP_DEBUG=false
```

  - Development:

```
APP_DEBUG=true
```

---

## Database

## Database Schema (ERD)

### Key Tables

**Users & Authentication:**

- users ` - User accounts, seller information`
- password*reset*tokens ` - Password reset tokens`
- sessions ` - User sessions` **E-Commerce:**
- products ` - Product catalog`
- product_variations ` - Product variations (size, color, etc.)`
- product_reviews ` - Product reviews`
- carts ` - Shopping carts`
- cart_items ` - Cart items`
- orders ` - Orders`
- order_items ` - Order line items`
- billing_details ` - Billing addresses`
- shipping_details ` - Shipping addresses`
- refunds ` - Refund requests`
- refund_images ` - Refund proof images`
- favourites ` - User favorites` **Services:**
- services ` - Service listings`
- service_reviews ` - Service reviews` **Content:**
- articles ` - Blog articles`
- videos ` - Video content`
- newsletters ` - Newsletter subscriptions` **System:**
- settings ` - System settings`
- migrations ` - Migration history`
- jobs ` - Queue jobs`
- cache ` - Cache storage`

## Key Relationships

**User Relationships:**

- User `hasMany` Order
- User `hasMany` Service
- User `hasMany` ProductReview
- User `hasMany` ServiceReview
- User `hasMany` Favourite
- User `hasMany` Refund **Order Relationships:**
- Order `belongsTo` User
- Order `hasMany` OrderItem
- Order `hasMany` Refund **Product Relationships:**
- Product `belongsTo` User `(seller)`
- Product `hasMany` ProductVariation
- Product `hasMany` ProductReview
- Product `hasMany` Favourite **Service Relationships:**
- Service `belongsTo` User `(seller)`
- Service `hasMany` ServiceReview

## Indexes

**Recommended Indexes:**

- users.email `- Unique index`
- orders.order_number `- Unique index`
- orders.user_id `- Index for user orders`
- orders.status `- Index for status filtering`
- products.slug `- Unique index`
- products.status `- Index for status filtering`
- services.slug `- Unique index`
- services.status `- Index for status filtering`

## Stored Procedures

**Current Status:** No stored procedures used **Note:** All database logic is handled through Eloquent ORM

---

## Migration History

## Migration Location

`database/migrations/`

### Key Migrations

**Initial Setup:**

- 0001*0101*000000create*users*table.php
- 0001*0101*000001create*cache*table.php
- 0001*0101*000002create*jobs*table.php **E-Commerce:**
- 2025*0718*084417create*products*table.php
- 2025*0718*095952create*product*variations_table.php
- 2025*0729*094332create*carts*table.php
- 2025*0730*094149create*orders*table.php
- 2025*0730*094155create*order*items_table.php **Services:**
- 2025*0717*000000create*services*table.php
- 2025*07112*create*service*reviews_table.php **Content:**
- 2025*0716*015359create*articles*table.php
- 2025*0716*135014create*videos*table.php
- 2025*0818*153741create*newsletters*table.php **User Features:**
- 2025*0101*000000create*favourites*table.php
- 2025*0101*000000create*refunds*table.php
- 2025*0730*100430create*billing*details_table.php
- 2025*0730*142618create*shipping*details_table.php

### Manual Database Steps

**If Migrations Fail:**

- Check database connection
- Verify table doesn't already exist
- Check for foreign key constraints
- Review migration files for errors

**Seeders:**

- DatabaseSeeder.php ` - Main seeder`
- AdminUserSeeder.php ` - Creates admin user`
- ProductSeeder.php ` - Sample products`
- SettingsSeeder.php ` - Default settings`

---

## Operations

## Common Tasks (Step-by-Step Guides)

## For Administrators

```
How to Add a New Product
```

- **Login to Admin Panel**

  - Go to

  `/admin/login`

  - Enter admin credentials

- **Navigate to Products**

  - Click "Products" in the sidebar

  - Click "Create Product" button

- **Fill in Product Details**

  - Product title (required)

  - Description

  - Price and sale price (if on sale)

  - Stock quantity

  - Category

  - Tags (comma-separated)

  - Upload product images

- **Add Variations (Optional)**

  - If product has sizes/colors, add variations

  - Each variation can have its own price and stock

- **Set Status**

  - Choose "Active" to make it visible

  - Choose "Inactive" to hide it temporarily

- **Save**

  - Click "Save" or "Create Product"

  - Product will appear in the shop

## How to Process a Refund

- **Navigate to Refunds**

  - Go to Admin Panel ? Refunds

  - Find the refund request

- **Review Refund Details**

  - Check order information

  - Review refund reason

  - Check uploaded proof/images

- **Approve or Reject**

  - **Approve:** Change status to "Approved" ? "Processing" ? "Completed"

  - **Reject:** Change status to "Rejected" and add reason

- **Update Order Status**

  - If approved, order status should be updated to "refunded"

## How to Approve a Seller

- **Navigate to Seller Requests**

  - Go to Admin Panel ? Seller Requests

  - Click "Pending" tab

- **Review Application**

  - Check business information

  - Review uploaded documents (ID, selfie, business registration)

  - Verify contact information

- **Approve or Reject**

  - **Approve:** Click "Approve" button

  - **Reject:** Click "Reject" and provide reason

- **Notify Seller**

  - System will notify seller of decision

  - Approved sellers can now list services

## How to Approve a Service

- **Navigate to Services**

  - Go to Admin Panel ? Services

  - Click "Pending" tab

- **Review Service**

  - Check service details

  - Review description and pricing

  - Check seller information

- **Approve or Reject**

  - **Approve:** Click "Approve" button

  - **Reject:** Click "Reject" and provide reason

- **Service Goes Live**

  - Approved services appear on marketplace

  - Rejected services can be edited and resubmitted

## How to Update Order Status

- **Find the Order**

  - Go to Admin Panel ? Orders

  - Search by order number or customer name

- **View Order Details**

  - Click on the order to see full details

  - Review items, shipping, and payment information

- **Update Status**

  - **Processing:** Order is being prepared

  - **Shipped:** Add tracking number and courier

  - **Delivered:** Mark as delivered

  - **Cancelled:** Cancel order (if needed)

- **Update Payment Status**

  - If payment received, update to "Paid"

  - If payment failed, update to "Failed"

## How to Check System Logs

- **Navigate to Logs**

  - Go to Admin Panel ? Logs

- **View Logs**

  - Logs are displayed with timestamps

  - Filter by log level (Error, Warning, Info, etc.)

  - Search for specific terms

- **Download Logs**

  - Click "Download" to save logs

  - Useful for troubleshooting

- **Clear Logs**

  - Click "Clear" to remove old logs

  - **Warning:** This cannot be undone!

## For Developers

How to Add a New Feature

- **Plan the Feature**

  - Define requirements

  - Identify affected files

  - Plan database changes (if needed)

- **Create Database Migration (if needed)**

  ` bash

  php artisan make:migration create*feature*table

  `

- **Create Model**

  ` bash

  php artisan make:model Feature

  `

- **Create Controller**

  ` bash

  php artisan make:controller FeatureController

  `

- **Add Routes**

  - Add routes in

  routes/web.php

- **Create Views**

  - Create Blade templates in

  resources/views/

- **Test**

  - Test the feature thoroughly

  - Check for errors in logs

- **Deploy**

  - Follow Deployment Process

## How to Debug an Issue

- **Check Logs**

```bash
tail -f storage/logs/laravel.log
```

- **Enable Debug Mode** (Development only)

  - Set

  APP_DEBUG=true `in` .env

- **Use Laravel Tinker**

```bash
php artisan tinker
# Test database queries
# Test model relationships
```

- **Check Database**

```bash
php artisan db:show
```

- **Clear Caches**

```bash
php artisan config:clear
php artisan route:clear
php artisan view:clear
```

## How to Deploy Updates

- **Test Locally**

  - Test all changes

  - Run migrations if needed

  - Check for errors

- **Commit Changes**

  ` bash

  git add .

  git commit -m "Description of changes"

  git push

  `

- **Deploy to Server**

  - Follow cPanel Deployment Guide

  - Or use your CI/CD pipeline

- **Run Migrations**

  ` bash

  php artisan migrate --force

  `

- **Clear Caches**

  ` bash

  php artisan config:cache

  php artisan route:cache

  php artisan view:cache

  `

- **Verify**

  - Check the site is working

  - Test critical features

  - Monitor logs for errors

  ---

### Admin Dashboard

**Access:** /admin/dashboard **Features:**

- Dashboard statistics
- Pending services count
- Pending sellers count
- Recent orders
- Quick actions

## Order Management

**View Orders:** `/admin/orders`

- Filter by status
- Search by order number
- Export orders
- View order details

**Update Order Status:**

- Navigate to order details
- Select new status
- Update payment status if needed
- Save changes

**Generate Invoice:**

- Open order details
- Click "View Invoice" or "Download Invoice"
- Invoice generated as PDF

## Product Management

**View Products:** /admin/products

- List all products
- Filter by status
- Search products
- Toggle featured status
- Toggle active status

**Create Product:**

- Click "Create Product"
- Fill in product details
- Add variations if needed
- Upload images
- Save

## Service Management

**View Services:** `/admin/services`

- List all services
- View pending services
- Approve/reject services
- Toggle service status

**Approve Service:**

- Navigate to pending services
- Review service details
- Click "Approve" or "Reject"
- Add rejection reason if rejecting

## User Management

**View Users:** `/admin/users`

- List all users
- Search users
- Verify users
- Suspend/activate users
- View user details

## Settings Management

**Access Settings:** `/admin/settings`

- Update system settings
- Reset to defaults
- Save changes

### Log Viewer

**Access Logs:** `/admin/logs`

- View Laravel logs
- Filter by log level
- Search logs
- Download logs
- Clear logs

---

## Maintenance Tasks

## Cron Jobs

**Required Cron Jobs:**

- **Auto-Cancel Pending Orders**

` bash

```
0     cd /path/to/project && php artisan orders:auto-cancel-pending >>
/dev/null 2>&1
```

`

- Frequency: Hourly

- Purpose: Cancel orders pending > 24 hours

- **Auto-Mark Delivered**

` bash

```
0 9   * cd /path/to/project && php artisan orders:auto-mark-delivered >>
/dev/null 2>&1
```

`

- Frequency: Daily at 9:00 AM

- Purpose: Mark shipped orders as delivered after 7 days

**Setup in cPanel:**

- Go to cPanel ? Cron Jobs
- Add new cron job
- Set schedule
- Enter command
- Save

## Log Rotation

**Laravel Logs:**

- Location:

`storage/logs/laravel.log`

- Rotation: Configure via server (logrotate)
- Manual cleanup:

`php artisan log:clear` (if command exists) **DomPDF Logs:**

- Location:

`storage/logs/dompdf.log`

- Manual cleanup: Delete old log entries

### SSL Renewal

**Current Setup:** Managed by hosting provider **Manual Renewal (if needed):**

- Access cPanel SSL/TLS section
- Follow provider instructions
- Verify certificate installation

### Data Cleanup

**Temporary Files:**

- Invoice files: Auto-deleted after email sending
- Old cache: Clear via

php artisan cache:clear **Old Orders:**

- Consider archiving orders older than 1 year
- Keep for accounting/legal purposes

**Log Files:**

- Rotate logs monthly
- Keep last 3 months of logs

## Regular Maintenance Checklist

**Weekly:**

- [ ] Check error logs
- [ ] Verify cron jobs are running
- [ ] Check disk space
- [ ] Review pending orders

**Monthly:**

- [ ] Review and rotate logs
- [ ] Check database size
- [ ] Verify backups
- [ ] Update dependencies (if needed)
- [ ] Review security updates

**Quarterly:**

- [ ] Full system backup
- [ ] Security audit
- [ ] Performance review
- [ ] Dependency updates

---

# Access & Accounts

## Credential Inventory

## Application Credentials

**Database:**

- **Location:**

.env `file`

- **Variables:**

DB*HOST* `,` *DB*DATABASE `,` DB*USERNAME* `,` *DB*PASSWORD

- **Access:** cPanel ? Databases ? MySQL Databases

**Payment Gateways:** **Stripe:**

- **Location:**

.env `file`

- **Variables:**

STRIPE*KEY* `,` *STRIPE*SECRET `,` STRIPE*WEBHOOK*SECRET

- **Access:** Stripe Dashboard ? Developers ? API keys
- **Owner:** [To be filled]

**ToyyibPay:**

- **Location:**

.env `file`

- **Variables:**

TOYYIBPAY*SECRET*KEY `,` TOYYIBPAY*CATEGORY*CODE

- **Access:** ToyyibPay Dashboard
- **Owner:** [To be filled]

**Email (SMTP):**

- **Location:**

.env `file`

- **Variables:**

MAIL*HOST* `,` *MAIL*USERNAME `,` MAIL_PASSWORD

- **Access:** cPanel ? Email Accounts
- **Owner:** [To be filled]

**Google OAuth:**

- **Location:**

.env `file`

- **Variables:**

GOOGLE*CLIENT*ID `,` GOOGLE*CLIENT*SECRET

- **Access:** Google Cloud Console

- **Owner:** [To be filled]

## Server Access

**cPanel:**

- **URL:** [To be filled]
- **Username:** [To be filled]
- **Password:** [To be filled - use password manager]
- **Owner:** [To be filled]

**FTP/SFTP:**

- **Host:** [To be filled]
- **Username:** [To be filled]
- **Password:** [To be filled - use password manager]
- **Port:** 21 (FTP) or 22 (SFTP)

**SSH (if available):**

- **Host:** [To be filled]
- **Username:** [To be filled]
- **Key/Password:** [To be filled - use password manager]

**Database (Direct Access):**

- **Host:** [To be filled]
- **Database:** [To be filled]
- **Username:** [To be filled]
- **Password:** [To be filled - use password manager]

## Third-Party Services

**Domain Registrar:**

- **Provider:** [To be filled]
- **Account:** [To be filled]
- **Owner:** [To be filled]

**Hosting Provider:**

- **Provider:** [To be filled]
- **Account:** [To be filled]
- **Owner:** [To be filled]

**Git Repository:**

- **Provider:** [To be filled]
- **Repository URL:** [To be filled]
- **Access:** [To be filled]

## Admin Accounts

**Super Admin:**

- **Email:** [To be filled]
- **Password:** [To be filled - use password manager]
- **Created By:** Seeder (

AdminUserSeeder.php ) **Note:** Admin accounts should be created via seeder or manually. Default admin can be created using: ` bash

```
php artisan db:seed --class=AdminUserSeeder
```

`

### Security Notes

- **Never commit**

  `.env` file to Git
- **Use password manager for all credentials**
- **Rotate passwords regularly**
- **Use 2FA where available**
- **Limit access to production credentials**
- **Document credential owners and access levels**

---

# Glossary of Terms

## Technical Terms Explained Simply

**API (Application Programming Interface)**

- A way for different software systems to talk to each other
- Example: Our system talks to Stripe's API to process payments

**Artisan**

- Laravel's command-line tool
- Used to run commands like

`php artisan migrate`

- Think of it as a helper that does tasks for you

**Blade**

- Laravel's templating engine
- Files that mix HTML and PHP code
- Located in

`resources/views/`

- Example:

`{{ $user->name }}` displays the user's name **Cache**

- Temporary storage for frequently used data
- Makes the system faster by storing data in memory
- Can be cleared with

`php artisan cache:clear` **Composer**

- PHP's package manager
- Downloads and manages PHP libraries
- Similar to npm for JavaScript

**Controller**

- Handles user requests and business logic
- Located in

`app/Http/Controllers/`

- Example: When user visits

`/shop` , `ProductController` handles it **Database Migration**

- A file that describes changes to the database structure
- Like a blueprint for creating/modifying tables
- Run with

`php artisan migrate` **Eloquent ORM**

- Laravel's way of talking to the database
- Makes database queries easier
- Example:

`User::find(1)` gets user with ID 1 **Environment Variables (.env)**

- Configuration file with sensitive settings
- Contains database passwords, API keys, etc.
- Never commit this file to Git!

**Framework**

- A foundation for building applications
- Laravel is a PHP framework
- Provides tools and structure so you don't start from scratch

**Git**

- Version control system
- Tracks changes to code
- Allows multiple developers to work together

**Middleware**

- Code that runs before/after requests
- Like a security guard checking requests
- Example: Authentication middleware checks if user is logged in

**Model**

- Represents a database table
- Located in

`app/Models/`

- Example:

`User` model represents the `users` table **MVC (Model-View-Controller)**

- A design pattern for organizing code
- **Model:** Database/data
- **View:** What user sees (HTML)
- **Controller:** Business logic

**Migration**

- See "Database Migration"

**npm**

- Node Package Manager
- Downloads and manages JavaScript/CSS libraries
- Similar to Composer for PHP

**ORM (Object-Relational Mapping)**

- See "Eloquent ORM"

**Route**

- Maps URLs to controller methods
- Defined in

`routes/web.php`

- `Example:`

/shop `?` ProductController@index **Seeder**

- Adds sample/test data to database
- Run with

php artisan db:seed

- Useful for testing

**Session**

- Temporary storage for user data
- Lasts while user is browsing
- Example: Shopping cart stored in session

**Storage Link**

- A symbolic link that makes uploaded files accessible
- Created with

php artisan storage:link

- Links

public/storage `to` storage/app/public **Webhook**

- A way for external services to notify our system
- Example: Stripe sends a webhook when payment is completed
- Our system listens and updates the order status

## Business Terms

**FPX (Financial Process Exchange)**

- Malaysian online banking payment system
- Allows customers to pay directly from their bank account
- Integrated via ToyyibPay

**Marketplace**

- A platform where multiple sellers can list their services
- Like Amazon or eBay, but for services

**OAuth**

- A way to login using external accounts (like Google)
- Users don't need to create a new account

**Payment Gateway**

- A service that processes payments
- Stripe and ToyyibPay are payment gateways
- Handles credit cards, bank transfers, etc.

**Refund**

- Returning money to a customer
- Can be requested by customer
- Must be approved by admin

**Seller**

- A user who has been approved to sell services
- Must go through application process
- Can list services on the marketplace

**Service**

- Something a seller offers (not a physical product)
- Examples: Web design, consulting, repairs
- Must be approved by admin before appearing

## Acronyms

**API** - Application Programming Interface **CSS** - Cascading Style Sheets (styling) **DB** - Database **FPX** - Financial Process Exchange **HTML** - HyperText Markup Language **HTTP** - HyperText Transfer Protocol **JS** - JavaScript **MVC** - Model-View-Controller **ORM** - Object-Relational Mapping **PHP** - PHP: Hypertext Preprocessor (programming language) **SMTP** - Simple Mail Transfer Protocol (email) **SQL** - Structured Query Language (database queries) **URL** - Uniform Resource Locator (web address)

## Common Phrases

**"Clear the cache"** - Remove temporary stored data **"Run migrations"** - Apply database structure changes **"Seed the database"** - Add sample/test data **"Deploy to production"** - Put changes on the live server **"Rollback"** - Undo changes and go back to previous version

---

# Master Handover Document

## Project Summary

**Project Name:** MyGooners **Type:** E-Commerce & Services Marketplace Platform **Framework:** Laravel 12.0 **Status:** Production Ready **Last Major Update:** January 2025

## Quick Links

## Documentation Files

- README.md `- Basic project information`
- CPANEL*DEPLOYMENT*GUIDE.md `- Deployment instructions`
- ADMIN*ORDERS*README.md `- Order management guide`
- REFUND*SYSTEM*README.md `- Refund system documentation`
- AUTO*CANCEL*FEATURE.md `- Auto-cancel feature`
- AUTO*DELIVERY*FEATURE.md `- Auto-delivery feature`
- STRIPE*PAYMENT*FIXES.md `- Stripe payment fixes`
- RETRY*PAYMENT*IMPROVEMENT.md `- Payment retry improvements`
- INVOICE*EMAIL*FEATURE.md `- Invoice email feature`
- ADMIN*INVOICE*FEATURE.md `- Admin invoice feature`
- ADMIN*LOGS*FEATURE.md `- Admin log viewer`
- SETTINGS*SYSTEM*README.md `- Settings system`
- FAVOURITES_FEATURE.md `- Favorites feature`
- PRODUCT*REPORTS*README.md `- Product reports`
- REQUEST*SYSTEM*README.md `- Request system`
- CANCEL*ORDER*README.md `- Cancel order feature`
- MARK*AS*DELIVERED_README.md `- Mark as delivered feature`
- ORDER*STATUS*CARDS_FEATURE.md `- Order status cards`
- RICH*TEXT*EDITOR_FEATURE.md `- Rich text editor`
- SOCIAL*MEDIA*EMBEDS_FEATURE.md `- Social media embeds`
- RESPONSIVE*NAVBAR*GUIDE.md `- Responsive navbar`
- TABLET*RESPONSIVE*DEMO.md `- Tablet responsive design`

## Key Directories

- app/Http/Controllers/ `- Application controllers`
- app/Services/ `- Business logic services`
- app/Models/ `- Eloquent models`
- routes/web.php `- Application routes`
- database/migrations/ `- Database migrations`
- resources/views/ `- Blade templates`

```
Configuration Files
```

- .env` - Environment configuration (DO NOT COMMIT)`
- config/app.php` - Application configuration`
- config/database.php` - Database configuration`
- config/services.php` - Third-party services`
- config/mail.php` - Email configuration

## Project Status

## Current State

? **Production Ready**

- Core features implemented
- Payment integration working
- Admin panel functional
- Automated order management

### Known Issues

- See Known Bugs & Limitations section
- Session timeout for payments (partially resolved)
- ToyyibPay cancellation limitation

## Next Steps

- Remove temporary routes (composer update route)
- Implement email queue system
- Add comprehensive testing
- Performance optimization
- Add more payment gateways (optional)

## Critical Information

### Deployment

- **Platform:** cPanel hosting
- **Domain:** mygooners.my
- **PHP Version:** 8.1+
- **Database:** MySQL/MariaDB

## Scheduled Tasks

- Auto-cancel pending orders: Hourly
- Auto-mark delivered: Daily at 9:00 AM

### Payment Gateways

- Stripe (credit cards)
- ToyyibPay (FPX, credit cards)

### Key Contacts

- **Project Owner:** [To be filled]
- **Technical Lead:** [To be filled]
- **Hosting Support:** [To be filled]
- **Payment Gateway Support:**

  - Stripe: support@stripe.com

  - ToyyibPay: [To be filled]

## Handover Checklist

### For Incoming Developer

- [ ] Review this documentation
- [ ] Set up local development environment
- [ ] Access to Git repository
- [ ] Access to cPanel/server
- [ ] Access to database
- [ ] Access to payment gateway dashboards
- [ ] Access to email accounts
- [ ] Review all credential inventory
- [ ] Understand deployment process
- [ ] Test local setup
- [ ] Review codebase structure
- [ ] Understand scheduled tasks
- [ ] Review known issues
- [ ] Access to monitoring/logs

### For Outgoing Developer

- [ ] Update credential inventory
- [ ] Document any recent changes
- [ ] Update known issues list
- [ ] Transfer access credentials
- [ ] Provide contact information for questions
- [ ] Schedule knowledge transfer session
- [ ] Document any undocumented features
- [ ] Update this documentation if needed

## Support Resources

## Laravel Documentation

- https://laravel.com/docs

## Payment Gateway Documentation

- Stripe: https://stripe.com/docs
- ToyyibPay: https://toyyibpay.com/documentation

## Project-Specific Documentation

- See Quick Links section above

## Final Notes

This documentation is a living document and should be updated as the project evolves. Regular updates are recommended when:

- New features are added
- Configuration changes are made
- New integrations are added
- Known issues are resolved
- Deployment process changes

**Last Review Date:** [To be filled] **Next Review Date:** [To be filled]

---

**End of Documentation**