Project Documentation: Implementation of Shipping Module in Laravel

---

Project Title:

Shipping Module Implementation for E-commerce Application Using Laravel and Shiprocket API

Objective:

To design and implement a scalable and efficient shipping module for an e-commerce application using Laravel, integrating shipping methods, dynamic cost calculation, and tracking functionality with the Shiprocket API.

Scope:

The shipping module will handle:

1. **Shipping Types:**
    - Self-shipped (Flat rate calculation)
    - Auto-shipped (Service provider-based calculation using APIs)
        - Service Timings
        - Hyper-local deliveries
        - Courier services
2. **Cost Calculation:** ○ Flat rate for self-shipped orders.
    - API-based dynamic calculation for auto-shipped orders.
3. **Tracking:**
    - Allow customers to track shipments in real-time.

Features:

1. **User Interface:**
    - Dropdowns and radio buttons for selecting shipping types. ○ Input fields for entering package details (e.g., weight, pickup and delivery postcode).
2. **Backend Integration:**
    - Integration with Shiprocket API for dynamic shipping cost calculation and tracking.
    - CRUD operations for managing shipping types and rates.
3. **API Endpoints:**
    - `/calculate-shipping` for shipping cost estimation.
    - `/track-shipment` for shipment tracking.

4. **Scalability:**
    - The module can handle multiple shipping providers if required in the future.

Tools and Technologies:

1. **Backend Framework:** Laravel 10
2. **Frontend Framework:** Blade Templating Engine
3. **Database:** MySQL
4. **HTTP Client:** Guzzle (for API requests)
5. **API Service Provider:** Shiprocket
6. **Testing Framework:** PHPUnit

Implementation Plan:

1. Requirement Analysis:

- Identify key functionalities required for the shipping module.
- Register and retrieve API credentials from Shiprocket.
- Understand Shiprocket API endpoints for:
    - Authentication ○ Serviceability ○ Tracking

2. Project Setup:

- Install Laravel framework using Composer.
- Configure the `.env` file for database and API credentials.
- Set up authentication middleware for secured API requests.

3. Database Design:

- Define tables for:
    - Shipping Types (id, type, method, flat_rate, created_at, updated_at) ○ Orders (id, shipping_type_id, weight, cost, tracking_id, created_at, updated_at)
    - Tracking Details (id, tracking_id, status, eta, created_at, updated_at)

4. API Integration:

- Authenticate with Shiprocket and store the token for reuse.
- Implement API calls for:
    - Cost calculation based on pickup and delivery postcodes and weight. ○ Real-time tracking of shipments.

5. Backend Logic:

- Create a `ShippingService` class to encapsulate API logic:
    - **Methods:**
        - `authenticate()`: Obtain API token.
        - `calculateShippingCost()`: Fetch cost details.
        - `trackShipment()`: Retrieve tracking information.
- Develop controllers for handling user requests and returning responses.

6. Frontend Development:

- Design a responsive user interface using Blade templates.
- Create forms for:
    - Selecting shipping type. o Entering package details.
    - Tracking shipments.
- Implement JavaScript to handle dynamic interactions (e.g., fetching cost via AJAX).

7. Testing:

- Unit Testing:
    - Write test cases for database models and API integration.
- Functional Testing:
    - Simulate user interactions with the UI and validate responses.

8. Deployment:

- Push the code to a version control system (e.g., GitHub).
- Deploy the application to a server (e.g., AWS, Heroku, Laravel Forge). Configure `.env` for production environment credentials.

9. Monitoring and Maintenance:

- Enable logging for API requests and responses.
- Monitor API token expiration and refresh tokens when required. Use Laravel Telescope to debug and monitor the application.

Timeline:

Task	Duration
Requirement Analysis	2 days
Project Setup	1 day
Database Design	3 days
API Integration	6 days
Backend Logic Implementation	6 days
Frontend Development	6 days
Testing	2 days
Deployment	1 day

Task	Duration
Total Time	**18 days**

Expected Outcomes:

1. A fully functional shipping module integrated with Shiprocket API.
2. Accurate cost calculation for self-shipped and auto-shipped orders.
3. Real-time tracking functionality accessible to customers.
4. Scalable architecture to accommodate future shipping providers or features.

Future Enhancements:

1. Add support for multiple shipping providers.
2. Enable bulk order shipping.
3. Integrate email/SMS notifications for tracking updates.
4. Implement a dashboard for admin to manage shipping providers and rates.

Conclusion:

The shipping module implementation in Laravel will significantly enhance the e-commerce application by providing dynamic cost calculation, multiple shipping options, and real-time tracking, improving the overall user experience.