# Laravel Backend Implementation for Delivery App System

I'll guide you through implementing the backend for your delivery app system using Laravel. This will be a step-by-step approach covering all the key features mentioned in your requirements.

## Step 1: Project Setup

First, let's set up the Laravel project and basic configuration:

```bash

composer create-project laravel/laravel delivery-app

cd delivery-app

```

Configure your `.env` file with database credentials:

```

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

DB\_PORT=3306

DB\_DATABASE=delivery\_app

DB\_USERNAME=root

DB\_PASSWORD=

```

## Step 2: Database Schema Design

Let's create migrations for all the necessary tables:

### Users Table (already exists, we'll modify it)

```bash

php artisan make:migration add\_role\_to\_users\_table

```

```php

// database/migrations/[timestamp]\_add\_role\_to\_users\_table.php

public function up()

{

Schema::table('users', function (Blueprint $table) {

$table->enum('role', ['admin', 'client', 'driver'])->default('client');

$table->string('phone')->nullable();

$table->string('avatar')->nullable();

$table->boolean('is\_verified')->default(false);

$table->string('verification\_token')->nullable();

});

}

```

### Drivers Table

```bash

php artisan make:model Driver -m

```

```php

// database/migrations/[timestamp]\_create\_drivers\_table.php

public function up()

{

Schema::create('drivers', function (Blueprint $table) {

$table->id();

$table->foreignId('user\_id')->constrained()->onDelete('cascade');

$table->enum('vehicle\_type', ['motorcycle', 'car', 'van', 'truck']);

$table->string('plate\_number');

$table->string('license\_number');

$table->enum('pricing\_model', ['fixed', 'per\_km']);

$table->decimal('fixed\_price', 10, 2)->nullable();

$table->decimal('price\_per\_km', 10, 2)->nullable();

$table->boolean('is\_approved')->default(false);

$table->decimal('rating', 3, 2)->default(0);

$table->integer('total\_ratings')->default(0);

$table->timestamps();

});

}

```

### Driver Working Areas

```bash

php artisan make:model DriverWorkingArea -m

```

```php

// database/migrations/[timestamp]\_create\_driver\_working\_areas\_table.php

public function up()

{

Schema::create('driver\_working\_areas', function (Blueprint $table) {

$table->id();

$table->foreignId('driver\_id')->constrained()->onDelete('cascade');

$table->string('city');

$table->timestamps();

});

}

```

### Deliveries Table

```bash

php artisan make:model Delivery -m

```

```php

// database/migrations/[timestamp]\_create\_deliveries\_table.php

public function up()

{

Schema::create('deliveries', function (Blueprint $table) {

$table->id();

$table->foreignId('client\_id')->constrained('users')->onDelete('cascade');

$table->foreignId('driver\_id')->nullable()->constrained('users')->onDelete('set null');

$table->string('pickup\_location');

$table->string('destination');

$table->decimal('distance', 10, 2)->comment('in kilometers');

$table->enum('package\_type', ['small', 'medium', 'large', 'extra\_large']);

$table->decimal('package\_weight', 10, 2)->comment('in kg');

$table->text('special\_instructions')->nullable();

$table->enum('status', ['pending', 'accepted', 'in\_progress', 'completed', 'cancelled'])->default('pending');

$table->decimal('price', 10, 2);

$table->enum('payment\_method', ['credit\_card', 'cryptocurrency', 'cash\_on\_delivery']);

$table->boolean('is\_paid')->default(false);

$table->timestamp('pickup\_time')->nullable();

$table->timestamp('delivery\_time')->nullable();

$table->timestamps();

});

}

```

### Payments Table

```bash

php artisan make:model Payment -m

```

```php

// database/migrations/[timestamp]\_create\_payments\_table.php

public function up()

{

Schema::create('payments', function (Blueprint $table) {

$table->id();

$table->foreignId('delivery\_id')->constrained()->onDelete('cascade');

$table->decimal('amount', 10, 2);

$table->string('currency')->default('USD');

$table->enum('method', ['credit\_card', 'cryptocurrency', 'cash\_on\_delivery']);

$table->string('transaction\_id')->nullable();

$table->enum('status', ['pending', 'completed', 'failed'])->default('pending');

$table->json('payment\_details')->nullable();

$table->timestamps();

});

}

```

### Reviews Table

```bash

php artisan make:model Review -m

```

```php

// database/migrations/[timestamp]\_create\_reviews\_table.php

public function up()

{

Schema::create('reviews', function (Blueprint $table) {

$table->id();

$table->foreignId('delivery\_id')->constrained()->onDelete('cascade');

$table->foreignId('client\_id')->constrained('users')->onDelete('cascade');

$table->foreignId('driver\_id')->constrained('users')->onDelete('cascade');

$table->integer('rating');

$table->text('comment')->nullable();

$table->timestamps();

});

}

```

### Messages Table

```bash

php artisan make:model Message -m

```

```php

// database/migrations/[timestamp]\_create\_messages\_table.php

public function up()

{

Schema::create('messages', function (Blueprint $table) {

$table->id();

$table->foreignId('delivery\_id')->constrained()->onDelete('cascade');

$table->foreignId('sender\_id')->constrained('users')->onDelete('cascade');

$table->foreignId('receiver\_id')->constrained('users')->onDelete('cascade');

$table->text('message');

$table->boolean('is\_read')->default(false);

$table->timestamps();

});

}

```

### Loyalty Points Table

```bash

php artisan make:model LoyaltyPoint -m

```

```php

// database/migrations/[timestamp]\_create\_loyalty\_points\_table.php

public function up()

{

Schema::create('loyalty\_points', function (Blueprint $table) {

$table->id();

$table->foreignId('driver\_id')->constrained('users')->onDelete('cascade');

$table->integer('points');

$table->decimal('kilometers', 10, 2);

$table->string('description');

$table->timestamps();

});

}

```

Run the migrations:

```bash

php artisan migrate

```

## Step 3: Model Relationships

Now let's set up the model relationships:

### User Model

```php

// app/Models/User.php

namespace App\Models;

use Illuminate\Contracts\Auth\MustVerifyEmail;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Notifications\Notifiable;

use Laravel\Sanctum\HasApiTokens;

class User extends Authenticatable

{

use HasApiTokens, HasFactory, Notifiable;

protected $fillable = [

'name',

'email',

'password',

'role',

'phone',

'avatar',

'is\_verified',

'verification\_token'

];

protected $hidden = [

'password',

'remember\_token',

];

protected $casts = [

'email\_verified\_at' => 'datetime',

'is\_verified' => 'boolean'

];

public function driverProfile()

{

return $this->hasOne(Driver::class);

}

public function workingAreas()

{

return $this->hasManyThrough(DriverWorkingArea::class, Driver::class);

}

public function clientDeliveries()

{

return $this->hasMany(Delivery::class, 'client\_id');

}

public function driverDeliveries()

{

return $this->hasMany(Delivery::class, 'driver\_id');

}

public function sentMessages()

{

return $this->hasMany(Message::class, 'sender\_id');

}

public function receivedMessages()

{

return $this->hasMany(Message::class, 'receiver\_id');

}

public function reviews()

{

return $this->hasMany(Review::class, 'driver\_id');

}

public function loyaltyPoints()

{

return $this->hasMany(LoyaltyPoint::class, 'driver\_id');

}

public function isAdmin()

{

return $this->role === 'admin';

}

public function isDriver()

{

return $this->role === 'driver';

}

public function isClient()

{

return $this->role === 'client';

}

}

```

### Driver Model

```php

// app/Models/Driver.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Driver extends Model

{

use HasFactory;

protected $fillable = [

'user\_id',

'vehicle\_type',

'plate\_number',

'license\_number',

'pricing\_model',

'fixed\_price',

'price\_per\_km',

'is\_approved',

'rating',

'total\_ratings'

];

public function user()

{

return $this->belongsTo(User::class);

}

public function workingAreas()

{

return $this->hasMany(DriverWorkingArea::class);

}

public function deliveries()

{

return $this->hasMany(Delivery::class, 'driver\_id');

}

public function reviews()

{

return $this->hasMany(Review::class, 'driver\_id');

}

public function loyaltyPoints()

{

return $this->hasMany(LoyaltyPoint::class, 'driver\_id');

}

}

```

### Delivery Model

```php

// app/Models/Delivery.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Delivery extends Model

{

use HasFactory;

protected $fillable = [

'client\_id',

'driver\_id',

'pickup\_location',

'destination',

'distance',

'package\_type',

'package\_weight',

'special\_instructions',

'status',

'price',

'payment\_method',

'is\_paid',

'pickup\_time',

'delivery\_time'

];

protected $casts = [

'pickup\_time' => 'datetime',

'delivery\_time' => 'datetime',

'is\_paid' => 'boolean'

];

public function client()

{

return $this->belongsTo(User::class, 'client\_id');

}

public function driver()

{

return $this->belongsTo(User::class, 'driver\_id');

}

public function payment()

{

return $this->hasOne(Payment::class);

}

public function review()

{

return $this->hasOne(Review::class);

}

public function messages()

{

return $this->hasMany(Message::class);

}

}

```

### Payment Model

```php

// app/Models/Payment.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Payment extends Model

{

use HasFactory;

protected $fillable = [

'delivery\_id',

'amount',

'currency',

'method',

'transaction\_id',

'status',

'payment\_details'

];

protected $casts = [

'payment\_details' => 'array'

];

public function delivery()

{

return $this->belongsTo(Delivery::class);

}

}

```

### Review Model

```php

// app/Models/Review.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Review extends Model

{

use HasFactory;

protected $fillable = [

'delivery\_id',

'client\_id',

'driver\_id',

'rating',

'comment'

];

public function delivery()

{

return $this->belongsTo(Delivery::class);

}

public function client()

{

return $this->belongsTo(User::class, 'client\_id');

}

public function driver()

{

return $this->belongsTo(User::class, 'driver\_id');

}

}

```

### Message Model

```php

// app/Models/Message.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class Message extends Model

{

use HasFactory;

protected $fillable = [

'delivery\_id',

'sender\_id',

'receiver\_id',

'message',

'is\_read'

];

protected $casts = [

'is\_read' => 'boolean'

];

public function delivery()

{

return $this->belongsTo(Delivery::class);

}

public function sender()

{

return $this->belongsTo(User::class, 'sender\_id');

}

public function receiver()

{

return $this->belongsTo(User::class, 'receiver\_id');

}

}

```

### LoyaltyPoint Model

```php

// app/Models/LoyaltyPoint.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

class LoyaltyPoint extends Model

{

use HasFactory;

protected $fillable = [

'driver\_id',

'points',

'kilometers',

'description'

];

public function driver()

{

return $this->belongsTo(User::class, 'driver\_id');

}

}

```

## Step 4: Authentication & User Registration

### Register Controller

```bash

php artisan make:controller Auth/RegisterController

```

```php

// app/Http/Controllers/Auth/RegisterController.php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use App\Models\User;

use App\Models\Driver;

use App\Models\DriverWorkingArea;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Hash;

use Illuminate\Support\Facades\Validator;

use Illuminate\Support\Str;

use Illuminate\Auth\Events\Registered;

class RegisterController extends Controller

{

public function register(Request $request)

{

$validator = Validator::make($request->all(), [

'name' => 'required|string|max:255',

'email' => 'required|string|email|max:255|unique:users',

'password' => 'required|string|min:8|confirmed',

'phone' => 'required|string|max:20',

'role' => 'required|in:client,driver',

'vehicle\_type' => 'required\_if:role,driver',

'plate\_number' => 'required\_if:role,driver',

'license\_number' => 'required\_if:role,driver',

'pricing\_model' => 'required\_if:role,driver',

'fixed\_price' => 'required\_if:pricing\_model,fixed|numeric|min:0',

'price\_per\_km' => 'required\_if:pricing\_model,per\_km|numeric|min:0',

'working\_areas' => 'required\_if:role,driver|array',

'working\_areas.\*' => 'string',

]);

if ($validator->fails()) {

return response()->json(['errors' => $validator->errors()], 422);

}

$verificationToken = Str::random(60);

$user = User::create([

'name' => $request->name,

'email' => $request->email,

'password' => Hash::make($request->password),

'phone' => $request->phone,

'role' => $request->role,

'verification\_token' => $verificationToken,

]);

if ($request->role === 'driver') {

$driver = Driver::create([

'user\_id' => $user->id,

'vehicle\_type' => $request->vehicle\_type,

'plate\_number' => $request->plate\_number,

'license\_number' => $request->license\_number,

'pricing\_model' => $request->pricing\_model,

'fixed\_price' => $request->fixed\_price,

'price\_per\_km' => $request->price\_per\_km,

'is\_approved' => false,

]);

foreach ($request->working\_areas as $area) {

DriverWorkingArea::create([

'driver\_id' => $driver->id,

'city' => $area,

]);

}

}

event(new Registered($user));

return response()->json([

'message' => 'User registered successfully. Please check your email for verification.',

'user' => $user,

], 201);

}

}

```

### Login Controller

```bash

php artisan make:controller Auth/LoginController

```

```php

// app/Http/Controllers/Auth/LoginController.php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

use Illuminate\Validation\ValidationException;

class LoginController extends Controller

{

public function login(Request $request)

{

$request->validate([

'email' => 'required|email',

'password' => 'required',

]);

if (Auth::attempt(['email' => $request->email, 'password' => $request->password])) {

$user = Auth::user();

if (!$user->is\_verified) {

Auth::logout();

throw ValidationException::withMessages([

'email' => ['Your account is not verified. Please check your email for verification link.']

]);

}

if ($user->isDriver() && !$user->driverProfile->is\_approved) {

Auth::logout();

throw ValidationException::withMessages([

'email' => ['Your driver account is pending approval. We will notify you once approved.']

]);

}

$token = $user->createToken('auth-token')->plainTextToken;

return response()->json([

'token' => $token,

'user' => $user,

'role' => $user->role,

]);

}

throw ValidationException::withMessages([

'email' => ['The provided credentials are incorrect.']

]);

}

public function logout(Request $request)

{

$request->user()->currentAccessToken()->delete();

return response()->json(['message' => 'Logged out successfully']);

}

}

```

### Email Verification

```bash

php artisan make:controller Auth/VerificationController

```

```php

// app/Http/Controllers/Auth/VerificationController.php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use App\Models\User;

use Illuminate\Http\Request;

class VerificationController extends Controller

{

public function verify($token)

{

$user = User::where('verification\_token', $token)->first();

if (!$user) {

return response()->json(['message' => 'Invalid verification token'], 404);

}

$user->update([

'is\_verified' => true,

'verification\_token' => null,

]);

return response()->json(['message' => 'Email verified successfully']);

}

public function resend(Request $request)

{

$request->validate(['email' => 'required|email']);

$user = User::where('email', $request->email)->first();

if (!$user) {

return response()->json(['message' => 'User not found'], 404);

}

if ($user->is\_verified) {

return response()->json(['message' => 'Email already verified'], 400);

}

$user->sendEmailVerificationNotification();

return response()->json(['message' => 'Verification link resent']);

}

}

```

## Step 5: API Routes

```php

// routes/api.php

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\Auth\RegisterController;

use App\Http\Controllers\Auth\LoginController;

use App\Http\Controllers\Auth\VerificationController;

// Authentication Routes

Route::post('/register', [RegisterController::class, 'register']);

Route::post('/login', [LoginController::class, 'login']);

Route::post('/logout', [LoginController::class, 'logout'])->middleware('auth:sanctum');

Route::get('/verify-email/{token}', [VerificationController::class, 'verify']);

Route::post('/resend-verification', [VerificationController::class, 'resend']);

// Protected Routes

Route::middleware(['auth:sanctum', 'verified'])->group(function () {

// User Profile

Route::get('/user', function (Request $request) {

return $request->user();

});

// Deliveries

Route::apiResource('deliveries', DeliveryController::class);

Route::post('/deliveries/{delivery}/assign-driver', [DeliveryController::class, 'assignDriver']);

Route::post('/deliveries/{delivery}/update-status', [DeliveryController::class, 'updateStatus']);

Route::post('/deliveries/{delivery}/rate', [DeliveryController::class, 'rateDriver']);

Route::post('/deliveries/{delivery}/messages', [MessageController::class, 'store']);

Route::get('/deliveries/{delivery}/messages', [MessageController::class, 'index']);

// Payments

Route::post('/deliveries/{delivery}/pay', [PaymentController::class, 'processPayment']);

// Admin Routes

Route::middleware(['admin'])->group(function () {

Route::get('/admin/drivers/pending', [AdminController::class, 'pendingDrivers']);

Route::post('/admin/drivers/{driver}/approve', [AdminController::class, 'approveDriver']);

Route::post('/admin/drivers/{driver}/reject', [AdminController::class, 'rejectDriver']);

Route::get('/admin/reports', [AdminController::class, 'generateReports']);

Route::post('/admin/loyalty-rules', [AdminController::class, 'setLoyaltyRules']);

});

// Driver Routes

Route::middleware(['driver'])->group(function () {

Route::get('/driver/deliveries', [DriverController::class, 'deliveries']);

Route::post('/driver/availability', [DriverController::class, 'setAvailability']);

Route::get('/driver/earnings', [DriverController::class, 'earnings']);

});

});

```

## Step 6: Delivery Controller

```bash

php artisan make:controller DeliveryController

```

```php

// app/Http/Controllers/DeliveryController.php

namespace App\Http\Controllers;

use App\Models\Delivery;

use App\Models\User;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Validator;

class DeliveryController extends Controller

{

public function index(Request $request)

{

$user = $request->user();

if ($user->isClient()) {

$deliveries = $user->clientDeliveries()->with(['driver', 'payment'])->get();

} elseif ($user->isDriver()) {

$deliveries = $user->driverDeliveries()->with(['client', 'payment'])->get();

} else {

$deliveries = Delivery::with(['client', 'driver', 'payment'])->get();

}

return response()->json($deliveries);

}

public function store(Request $request)

{

$validator = Validator::make($request->all(), [

'pickup\_location' => 'required|string',

'destination' => 'required|string',

'package\_type' => 'required|in:small,medium,large,extra\_large',

'package\_weight' => 'required|numeric|min:0.1',

'special\_instructions' => 'nullable|string',

'auto\_assign' => 'required|boolean',

'driver\_id' => 'required\_if:auto\_assign,false|exists:users,id',

]);

if ($validator->fails()) {

return response()->json(['errors' => $validator->errors()], 422);

}

// Calculate distance (in a real app, you'd use a mapping API)

$distance = $this->calculateDistance($request->pickup\_location, $request->destination);

// Calculate price

$price = $this->calculatePrice(

$distance,

$request->package\_type,

$request->package\_weight,

$request->auto\_assign ? null : $request->driver\_id

);

$delivery = Delivery::create([

'client\_id' => $request->user()->id,

'driver\_id' => $request->auto\_assign ? $this->findAvailableDriver($request->pickup\_location) : $request->driver\_id,

'pickup\_location' => $request->pickup\_location,

'destination' => $request->destination,

'distance' => $distance,

'package\_type' => $request->package\_type,

'package\_weight' => $request->package\_weight,

'special\_instructions' => $request->special\_instructions,

'status' => 'pending',

'price' => $price,

'payment\_method' => 'cash\_on\_delivery', // Default, can be changed later

]);

return response()->json($delivery, 201);

}

public function show(Delivery $delivery)

{

$delivery->load(['client', 'driver', 'payment', 'messages', 'review']);

return response()->json($delivery);

}

public function update(Request $request, Delivery $delivery)

{

$validator = Validator::make($request->all(), [

'pickup\_location' => 'sometimes|required|string',

'destination' => 'sometimes|required|string',

'package\_type' => 'sometimes|required|in:small,medium,large,extra\_large',

'package\_weight' => 'sometimes|required|numeric|min:0.1',

'special\_instructions' => 'nullable|string',

]);

if ($validator->fails()) {

return response()->json(['errors' => $validator->errors()], 422);

}

$delivery->update($request->only([

'pickup\_location',

'destination',

'package\_type',

'package\_weight',

'special\_instructions',

]));

return response()->json($delivery);

}

public function destroy(Delivery $delivery)

{

$delivery->delete();

return response()->json(null, 204);

}

public function assignDriver(Request $request, Delivery $delivery)

{

$validator = Validator::make($request->all(), [

'driver\_id' => 'required|exists:users,id',

]);

if ($validator->fails()) {

return response()->json(['errors' => $validator->errors()], 422);

}

$delivery->update([

'driver\_id' => $request->driver\_id,

'status' => 'accepted',

]);

// Notify driver about new assignment

// $driver = User::find($request->driver\_id);

// $driver->notify(new NewDeliveryAssignment($delivery));

return response()->json($delivery);

}

public function updateStatus(Request $request, Delivery $delivery)

{

$validator = Validator::make($request->all(), [

'status' => 'required|in:accepted,in\_progress,completed,cancelled',

]);

if ($validator->fails()) {

return response()->json(['errors' => $validator->errors()], 422);

}

$delivery->update([

'status' => $request->status,

'delivery\_time' => $request->status === 'completed' ? now() : null,

]);

// Notify client about status change

// $delivery->client->notify(new DeliveryStatusUpdated($delivery));

return response()->json($delivery);

}

public function rateDriver(Request $request, Delivery $delivery)

{

$validator = Validator::make($request->all(), [

'rating' => 'required|integer|min:1|max:5',

'comment' => 'nullable|string',

]);

if ($validator->fails()) {

return response()->json(['errors' => $validator->errors()], 422);

}

if ($delivery->client\_id !== $request->user()->id) {

return response()->json(['message' => 'Unauthorized'], 403);

}

if ($delivery->status !== 'completed') {

return response()->json(['message' => 'Delivery must be completed before rating'], 400);

}

$review = $delivery->review()->create([

'client\_id' => $request->user()->id,

'driver\_id' => $delivery->driver\_id,

'rating' => $request->rating,

'comment' => $request->comment,

]);

// Update driver's average rating

$driver = $delivery->driver;

$driver->driverProfile->update([

'total\_ratings' => $driver->driverProfile->total\_ratings + 1,

'rating' => ($driver->driverProfile->rating \* $driver->driverProfile->total\_ratings + $request->rating) / ($driver->driverProfile->total\_ratings + 1),

]);

return response()->json($review, 201);

}

private function calculateDistance($pickup, $destination)

{

// In a real app, you'd use Google Maps API or similar

return rand(1, 50); // Random distance for demo

}

private function calculatePrice($distance, $packageType, $packageWeight, $driverId = null)

{

$basePrice = 5.00;

$typeMultipliers = [

'small' => 1.0,

'medium' => 1.2,

'large' => 1.5,

'extra\_large' => 2.0,

];

$weightMultiplier = min(1 + ($packageWeight / 10), 3); // Max 3x multiplier

if ($driverId) {

$driver = User::find($driverId)->driverProfile;

if ($driver->pricing\_model === 'fixed') {

return $driver->fixed\_price \* $typeMultipliers[$packageType] \* $weightMultiplier;

} else {

return $basePrice + ($distance \* $driver->price\_per\_km) \* $typeMultipliers[$packageType] \* $weightMultiplier;

}

}

// Default pricing if no specific driver

return $basePrice + ($distance \* 1.20) \* $typeMultipliers[$packageType] \* $weightMultiplier;

}

private function findAvailableDriver($location)

{

// In a real app, you'd find the nearest available driver

$driver = User::where('role', 'driver')

->whereHas('driverProfile', function ($query) {

$query->where('is\_approved', true);

})

->whereHas('workingAreas', function ($query) use ($location) {

$query->where('city', 'like', "%$location%");

})

->inRandomOrder()

->first();

return $driver ? $driver->id : null;

}

}

```

## Step 7: Payment Controller

```bash

php artisan make:controller PaymentController

```

```php

// app/Http/Controllers/PaymentController.php

namespace App\Http\Controllers;

use App\Models\Delivery;

use App\Models\Payment;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Validator;

class PaymentController extends Controller

{

public function processPayment(Request $request, Delivery $delivery)

{

$validator = Validator::make($request->all(), [

'payment\_method' => 'required|in:credit\_card,cryptocurrency,cash\_on\_delivery',

'card\_number' => 'required\_if:payment\_method,credit\_card',

'card\_expiry' => 'required\_if:payment\_method,credit\_card',

'card\_cvv' => 'required\_if:payment\_method,credit\_card',

'cryptocurrency' => 'required\_if:payment\_method,cryptocurrency|in:bitcoin,ethereum,litecoin',

]);

if ($validator->fails()) {

return response()->json(['errors' => $validator->errors()], 422);

}

if ($delivery->client\_id !== $request->user()->id) {

return response()->json(['message' => 'Unauthorized'], 403);

}

if ($delivery->is\_paid) {

return response()->json(['message' => 'Delivery already paid'], 400);

}

$paymentDetails = [];

$transactionId = null;

$status = 'pending';

if ($request->payment\_method === 'credit\_card') {

// In a real app, you'd process the payment with Stripe/PayPal/etc.

$paymentDetails = [

'card\_last4' => substr($request->card\_number, -4),

'card\_brand' => $this->detectCardType($request->card\_number),

];

$transactionId = 'cc\_' . uniqid();

$status = 'completed';

} elseif ($request->payment\_method === 'cryptocurrency') {

$paymentDetails = [

'cryptocurrency' => $request->cryptocurrency,

'amount' => $this->convertToCrypto($delivery->price, $request->cryptocurrency),

'wallet\_address' => config('app.crypto\_wallet\_address'),

];

$transactionId = 'crypto\_' . uniqid();

$status = 'pending'; // Would wait for blockchain confirmation

}

$payment = Payment::create([

'delivery\_id' => $delivery->id,

'amount' => $delivery->price,

'method' => $request->payment\_method,

'transaction\_id' => $transactionId,

'status' => $status,

'payment\_details' => $paymentDetails,

]);

if ($status === 'completed') {

$delivery->update([

'is\_paid' => true,

'payment\_method' => $request->payment\_method,

]);

}

return response()->json($payment, 201);

}

private function detectCardType($cardNumber)

{

$firstDigit = substr($cardNumber, 0, 1);

switch ($firstDigit) {

case '3': return 'amex';

case '4': return 'visa';

case '5': return 'mastercard';

case '6': return 'discover';

default: return 'unknown';

}

}

private function convertToCrypto($amount, $crypto)

{

// In a real app, you'd get current rates from an API

$rates = [

'bitcoin' => 28450.00,

'ethereum' => 1780.00,

'litecoin' => 85.00,

];

return $amount / $rates[$crypto];

}

}

```

## Step 8: Message Controller

```bash

php artisan make:controller MessageController

```

```php

// app/Http/Controllers/MessageController.php

namespace App\Http\Controllers;

use App\Models\Delivery;

use App\Models\Message;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Validator;

class MessageController extends Controller

{

public function index(Delivery $delivery)

{

$messages = $delivery->messages()

->with(['sender', 'receiver'])

->orderBy('created\_at', 'asc')

->get();

return response()->json($messages);

}

public function store(Request $request, Delivery $delivery)

{

$validator = Validator::make($request->all(), [

'message' => 'required|string|max:1000',

]);

if ($validator->fails()) {

return response()->json(['errors' => $validator->errors()], 422);

}

$user = $request->user();

$receiverId = $user->id === $delivery->client\_id ? $delivery->driver\_id : $delivery->client\_id;

$message = Message::create([

'delivery\_id' => $delivery->id,

'sender\_id' => $user->id,

'receiver\_id' => $receiverId,

'message' => $request->message,

]);

// In a real app, you'd send a real-time notification

// event(new NewMessage($message));

return response()->json($message, 201);

}

}

```

## Step 9: Admin Controller

```bash

php artisan make:controller AdminController