Next-Venture Laravel 12 — Project Skeleton

This document contains a ready-to-paste Laravel project skeleton for your assessment. Drop the files into `/Users/dopesoul/Data/Work/web_apps/htdocs/next-venture` (or use it as reference) and run `composer install` + `php artisan migrate`.

> Files are grouped by path. For each file: copy the file path then the file contents into your project.

---

1) .env (example)

```
APP_NAME=NextVenture
APP_ENV=local
APP_KEY=
APP_DEBUG=true
APP_URL=http://next-venture.local:8080

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=next_venture
DB_USERNAME=root
DB_PASSWORD=

QUEUE_CONNECTION=redis
CACHE_STORE=redis
REDIS_CLIENT=phpredis
REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_MAILER=log
MAIL_FROM_ADDRESS="hello@nextventure.local"
MAIL_FROM_NAME="${APP_NAME}"
```

---

2) routes/web.php

```php
<?php
use Illuminate\Support\Facades\Route;

Route::get('/', function () {
    return response('NextVenture Laravel 12 Project is Running!');
});
```

3) app/Console/Commands/ImportOrders.php

```php
<?php
namespace App\Console\Commands;

use Illuminate\Console\Command;
use App\Jobs\ProcessOrderJob;

class ImportOrders extends Command
{
    protected $signature = 'orders:import {file}';
    protected $description = 'Import orders from a CSV file (header
required)';

    public function handle()
    {
        $path = $this->argument('file');

        if (!file_exists($path)) {
            $this->error("File not found: {$path}");
            return 1;
        }

        $handle = fopen($path, 'r');
        if ($handle === false) {
            $this->error('Unable to open file');
            return 1;
        }

        $header = fgetcsv($handle);
        if ($header === false) {
            $this->error('CSV file empty or header missing');
            fclose($handle);
            return 1;
        }

        $count = 0;
        while (($row = fgetcsv($handle)) !== false) {
            $data = array_combine($header, $row);
            // Dispatch the job to process this order.
            ProcessOrderJob::dispatch($data)->onQueue('orders');
            $count++;
        }

        fclose($handle);
        $this->info("Queued {$count} orders for processing.");
        return 0;
```

```
    }
}
```

---

4) app/Jobs/ProcessOrderJob.php

```php
<?php
namespace App\Jobs;

use App\Models\Order;
use App\Models\OrderItem;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class ProcessOrderJob implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public $orderData;

    public function __construct(array $orderData)
    {
        $this->orderData = $orderData;
    }

    public function handle()
    {
        // Basic required fields: external_id, customer_id, total_cents
        if (empty($this->orderData['external_id'])) return;

        // Idempotent create: ensure we don't create duplicate orders
        $order = Order::firstOrCreate(
            ['external_id' => $this->orderData['external_id']],
            [
                'customer_id' => $this->orderData['customer_id'] ?? 0,
                'total_cents' => (int) ($this->orderData['total_cents'] ??
0),
                'currency' => $this->orderData['currency'] ?? 'USD',
                'status' => 'pending',
                'metadata' => $this->orderData,
            ]
        );

        // Optionally create order items if present (CSV column pattern:
item_1_sku,item_1_qty,etc.)
        // For the skeleton we skip detailed parsing.
```

```
        // Chain: ReserveStock -> SimulatePayment (which will trigger
callback)
        ReserveStockJob::withChain([
            new SimulatePaymentJob($order->id),
        ])->dispatch($order->id)->onQueue('reservations');
    }
}
```

---

5) app/Jobs/ReserveStockJob.php

```
<?php
namespace App\Jobs;

use App\Models\Order;
use App\Models\Product;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use Illuminate\Support\Facades\DB;

class ReserveStockJob implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public $orderId;

    public function __construct($orderId)
    {
        $this->orderId = $orderId;
    }

    public function handle()
    {
        $order = Order::find($this->orderId);
        if (!$order) return;
        if ($order->status !== 'pending') return;

        // Simplified: assume single product and sufficient stock.
        // In production, iterate items and lock product rows.
        DB::transaction(function () use ($order) {
            // Example: reduce stock for first matching product (demo only)
            $item = $order->items()->first();
            if (!$item) {
                // nothing to reserve — mark reserved and continue
                $order->update(['status' => 'reserved', 'reserved_at' =>
```

```
now()]);
                return;
            }

            $product = Product::where('id', $item->product_id)-
>lockForUpdate()->first();
            if (!$product || $product->stock < $item->quantity) {
                $order->update(['status' => 'failed', 'failed_at' => now()]);
                // dispatch notification
                SendOrderNotificationJob::dispatch($order->id, 'log')-
>onQueue('notifications');
                return;
            }

            $product->stock -= $item->quantity;
            $product->save();

            $order->update(['status' => 'reserved', 'reserved_at' => now()]);
        });
    }
}
```

---

6) app/Jobs/SimulatePaymentJob.php

```
<?php
namespace App\Jobs;

use App\Models\Order;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class SimulatePaymentJob implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public $orderId;

    public function __construct($orderId)
    {
        $this->orderId = $orderId;
    }

    public function handle()
    {
        $order = Order::find($this->orderId);
```

```
        if (!$order) return;

        // Simulate async payment by enqueuing a callback job with random
result
        $success = rand(0, 100) > 10; // 90% success for simulation
        // Pass a payment_id for idempotency in real systems
        PaymentCallbackJob::dispatch($order->id, $success ? 'success' :
'failed')->onQueue('payments');
    }
}
```

---

7) app/Jobs/PaymentCallbackJob.php

```
<?php
namespace App\Jobs;

use App\Models\Order;
use App\Services\KpiService;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class PaymentCallbackJob implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public $orderId;
    public $result; // 'success' or 'failed'

    public function __construct($orderId, $result)
    {
        $this->orderId = $orderId;
        $this->result = $result;
    }

    public function handle(KpiService $kpi)
    {
        $order = Order::find($this->orderId);
        if (!$order) return;

        // Idempotency: ignore if already completed or failed
        if (in_array($order->status, ['completed','failed','refunded']))
return;

        if ($this->result === 'success') {
            $order->update(['status' => 'completed', 'completed_at' =>
```

```
now()]);
            // Update KPIs and leaderboard
            $kpi->recordOrderCompleted($order);
            // Send notification (async)
            SendOrderNotificationJob::dispatch($order->id, 'log')-
>onQueue('notifications');
        } else {
            // Rollback reservation if any (simple version)
            foreach ($order->items as $item) {
                $product = $item->product;
                if ($product) {
                    $product->increment('stock', $item->quantity);
                }
            }
            $order->update(['status' => 'failed', 'failed_at' => now()]);
            SendOrderNotificationJob::dispatch($order->id, 'log')-
>onQueue('notifications');
        }
    }
}
```

---

8) app/Jobs/SendOrderNotificationJob.php

```
<?php
namespace App\Jobs;

use App\Models\Order;
use App\Models\NotificationHistory;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class SendOrderNotificationJob implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public $orderId;
    public $channel; // 'email' or 'log'

    public function __construct($orderId, $channel = 'log')
    {
        $this->orderId = $orderId;
        $this->channel = $channel;
    }

    public function handle()
```

```
    {
        $order = Order::with('items')->find($this->orderId);
        if (!$order) return;

        $payload = [
            'order_id' => $order->id,
            'customer_id' => $order->customer_id,
            'status' => $order->status,
            'total' => $order->total_cents,
        ];

        // For local development we just log; you can replace with
Mail::to(...)->queue(...)
        logger()->info('Order notification', $payload);

        NotificationHistory::create([
            'order_id' => $order->id,
            'customer_id' => $order->customer_id,
            'channel' => $this->channel,
            'payload' => $payload,
            'status' => 'sent',
            'sent_at' => now(),
        ]);
    }
}
```

---

9) app/Jobs/ProcessRefundJob.php

```
<?php
namespace App\Jobs;

use App\Models\Refund;
use App\Models\Order;
use App\Services\KpiService;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use Illuminate\Support\Facades\DB;

class ProcessRefundJob implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public $refundId;

    public function __construct($refundId)
```

```
    {
        $this->refundId = $refundId;
    }

    public function handle(KpiService $kpi)
    {
        $refund = Refund::find($this->refundId);
        if (!$refund) return;

        // Idempotency: if already processed, exit
        if ($refund->status === 'processed') return;

        DB::transaction(function () use ($refund, $kpi) {
            $order = Order::lockForUpdate()->find($refund->order_id);
            if (!$order) {
                $refund->update(['status' => 'failed']);
                return;
            }

            // Calculate refundable amount guard: do not refund more than
order total
            $refundableRemaining = max(0, $order->total_cents - ($order-
>refunded_cents ?? 0));
            $amount = min($refund->amount_cents, $refundableRemaining);
            if ($amount <= 0) {
                $refund->update(['status' => 'failed']);
                return;
            }

            // Apply refund: update order refunded_cents and possibly status
            $order->refunded_cents = ($order->refunded_cents ?? 0) + $amount;
            if ($order->refunded_cents >= $order->total_cents) {
                $order->status = 'refunded';
            }
            $order->save();

            // Mark refund processed
            $refund->update(['status' => 'processed']);

            // Update KPIs (subtract revenue) and leaderboard
            $kpi->applyRefund($order, $amount);
        });
    }
}
```

---

10) app/Services/KpiService.php

```php
<?php
namespace App\Services;

use Illuminate\Support\Facades\Redis;
use App\Models\Order;

class KpiService
{
    public function recordOrderCompleted(Order $order)
    {
        $date = $order->created_at ? $order->created_at->toDateString() :
now()->toDateString();
        Redis::incrby("kpi:revenue:{$date}", $order->total_cents);
        Redis::incr("kpi:orders:{$date}");
        Redis::zincrby("leaderboard:customers", $order->total_cents,
"customer:{$order->customer_id}");
    }

    public function applyRefund(Order $order, int $amountCents)
    {
        $date = $order->created_at ? $order->created_at->toDateString() :
now()->toDateString();
        Redis::decrby("kpi:revenue:{$date}", $amountCents);
        Redis::zincrby("leaderboard:customers", -$amountCents, "customer:
{$order->customer_id}");
    }
}
```

---

11) app/Models/Order.php

```php
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Order extends Model
{
    use HasFactory;

    protected $fillable = [
        'external_id', 'customer_id', 'total_cents', 'currency', 'status',
'metadata', 'refunded_cents'
    ];
```

```
    protected $casts = [
        'metadata' => 'array',
    ];

    public function items()
    {
        return $this->hasMany(OrderItem::class);
    }
}
```

---

12) app/Models/OrderItem.php

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class OrderItem extends Model
{
    use HasFactory;

    protected $fillable = ['order_id', 'product_id', 'quantity',
'price_cents'];

    public function order()
    {
        return $this->belongsTo(Order::class);
    }

    public function product()
    {
        return $this->belongsTo(Product::class);
    }
}
```

---

13) app/Models/Product.php

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Product extends Model
```

```
{
    use HasFactory;

    protected $fillable = ['name', 'stock', 'price_cents'];
}
```

---

14) app/Models/NotificationHistory.php

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class NotificationHistory extends Model
{
    use HasFactory;

    protected $table = 'notification_history';

    protected $fillable = ['order_id', 'customer_id', 'channel', 'payload',
'status', 'sent_at'];

    protected $casts = [
        'payload' => 'array',
        'sent_at' => 'datetime',
    ];
}
```

---

15) app/Models/Refund.php

```
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Refund extends Model
{
    use HasFactory;

    protected $fillable = ['order_id', 'amount_cents', 'status',
'external_id'];
}
```

16) app/Models/DailyKpi.php

```php
<?php
namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class DailyKpi extends Model
{
    use HasFactory;

    protected $fillable = ['date', 'revenue_cents', 'orders_count',
'avg_order_value'];

    protected $casts = [
        'date' => 'date',
    ];
}
```

17) database/migrations/ 2025_10_13_000001_create_orders_table.php

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up()
    {
        Schema::create('orders', function (Blueprint $table) {
            $table->id();
            $table->string('external_id')->unique();
            $table->unsignedBigInteger('customer_id')->index();
            $table->integer('total_cents');
            $table->string('currency', 3)->default('USD');
            $table->enum('status',
['pending','reserved','payment_pending','completed','failed','refunded'])-
>default('pending');
            $table->integer('refunded_cents')->default(0);
            $table->timestamp('reserved_at')->nullable();
            $table->timestamp('completed_at')->nullable();
            $table->timestamp('failed_at')->nullable();
            $table->json('metadata')->nullable();
```

```
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('orders');
    }
};
```

---

18) database/migrations/ 2025_10_13_000002_create_order_items_table.php

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up()
    {
        Schema::create('order_items', function (Blueprint $table) {
            $table->id();
            $table->foreignId('order_id')->constrained('orders')-
>onDelete('cascade');
            $table->foreignId('product_id')->constrained('products');
            $table->integer('quantity');
            $table->integer('price_cents');
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('order_items');
    }
};
```

---

19) database/migrations/ 2025_10_13_000003_create_products_table.php

```
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
```

```php
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up()
    {
        Schema::create('products', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->integer('stock')->default(0);
            $table->integer('price_cents')->default(0);
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('products');
    }
};
```

---

20) database/migrations/ 2025_10_13_000004_create_notification_history_table.php

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up()
    {
        Schema::create('notification_history', function (Blueprint $table) {
            $table->id();
            $table->foreignId('order_id')->constrained('orders')-
>onDelete('cascade');
            $table->unsignedBigInteger('customer_id');
            $table->string('channel');
            $table->json('payload');
            $table->enum('status', ['sent','failed'])->default('sent');
            $table->timestamp('sent_at')->nullable();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('notification_history');
```

```
    }
};
```

---

21) database/migrations/ 2025_10_13_000005_create_refunds_table.php

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up()
    {
        Schema::create('refunds', function (Blueprint $table) {
            $table->id();
            $table->foreignId('order_id')->constrained('orders');
            $table->integer('amount_cents');
            $table->enum('status', ['pending','processed','failed'])->default('pending');
            $table->string('external_id')->unique();
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('refunds');
    }
};
```

---

22) database/migrations/ 2025_10_13_000006_create_daily_kpis_table.php

```php
<?php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration {
    public function up()
    {
        Schema::create('daily_kpis', function (Blueprint $table) {
            $table->id();
```

```
            $table->date('date')->unique();
            $table->integer('revenue_cents')->default(0);
            $table->integer('orders_count')->default(0);
            $table->decimal('avg_order_value', 12, 2)->default(0);
            $table->timestamps();
        });
    }

    public function down()
    {
        Schema::dropIfExists('daily_kpis');
    }
};
```

---

23) README snippet — Quick setup

```
# Quick local setup (macOS, Laravel 12)

# 1) Install deps
composer install
cp .env.example .env
php artisan key:generate

# 2) configure DB in .env (MySQL)
# create DB: CREATE DATABASE next_venture;

# 3) run migrations
php artisan migrate

# 4) start Redis
brew services start redis

# 5) run Horizon in dev
php artisan horizon

# 6) run queue worker if not using Horizon
php artisan queue:work

# 7) test import (place sample orders.csv in project root)
php artisan orders:import orders.csv

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

---

24) Notes & next steps

- This skeleton focuses on **correct structure and idempotency patterns**. Business logic is intentionally simple and annotated so you can extend it as needed.
- Replace simulation stubs (payment randomizer, simplistic stock reservation) with production logic later (Redis Lua for reservations, real payment gateway/webhook handling, email channel).
- Add unit/integration tests for: import command, job chaining, refund idempotency, KPI updates.

If you want, I can now: - Produce the individual files as real files inside your project (I can paste exact `cp` / `tee` commands), or - Generate a single archive (zip) you can download — or - Add a few **example unit tests** and a small `orders.csv` example.

Which would you like next?