BMIT3173 Integrative Programming

ASSIGNMENT 202509

Student Name	:	Ong Yi Xin
Student ID	:	24WMR09097
Programme	:	RSD3
Tutorial Group	:	G5
System Title	:	TARUMT Event Management System
Modules	:	Payments Module

Plagiarism Statement Form

I, Name (Block Capitals) __ONG YI XIN______Student ID____24WMR09097__ Programme ___RSD___Tutorial Group ___5___confirm that the submitted work are all my own work and is in my own words.

I ONG YI XIN acknowledge the use of AI generative technology.

Signature: _____-____

Date: _____19/12/2025______

Table of Contents

1. Introduction to the System

The TARUMT Event Management System is a centralized web-based platform designed to optimize event organization and participation processes within the TARUMT community. The system comprises six core modules which are Users and Membership, Events and Helpers, Payments and Feedback. The Users module manages authentication, role assignment and access control for administrators, organizers and members. The Membership module handles organizer membership applications, approvals and credit tracking. The Events module enables organizers to create, manage and monitor events throughout their lifecycle. The Helpers module facilitates the creation, application and assignment management of helper positions for events. The Payments module handles event-related transactions, deposits, penalties, refunds and receipt generation. The Feedback module allows participants to submit ratings and comments, supporting event quality assurance and continuous improvement.

2. Module Description

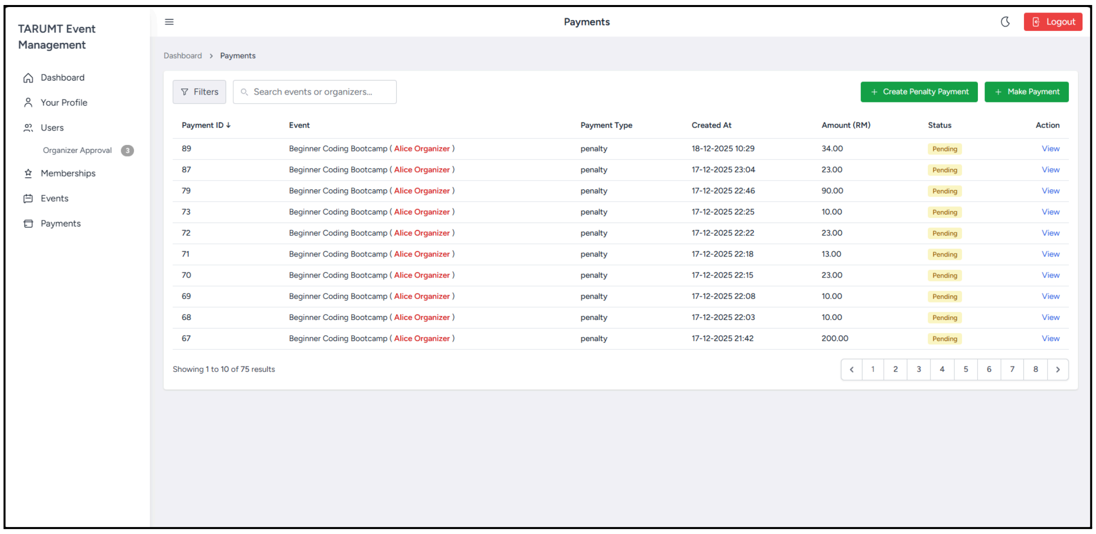
The Payment Module manages all financial transactions within the TARUMT Event Management System, ensuring secure, transparent and traceable processing of event-related payments. This module supports multiple payment scenarios, including event deposits, penalties, refunds and receipt generation. It controls access to related functions by validating payment statuses such as paid, pending, overdue and other business rules. By maintaining detailed payment records and logs, the module assists administrators in monitoring financial activities, resolving disputes and ensuring compliance. Overall, the Payment Module plays a critical role in upholding financial accountability and safeguarding the smooth operation of activities within the system.

1. **<u>View Payment List (Admin & Organizer)</u>**

   The payment list feature enables administrators and organizers to view payment records within the system. Administrators can access payment records for all activities and organizers, achieving full-process visualization for financial monitoring and management. Organizers are restricted to viewing payment records related to their own activities, ensuring data privacy and ownership control.

   The payment list supports search, filter and sort functions to help users quickly locate relevant payment records. Payments can be filtered by status, payment type, amount range and creation date, enabling efficient review of payment history and identification of outstanding or completed payments. This feature provides a clear overview of payment activities, offering administrators and organizers efficient financial tracking support.

   Class path: /resources/views/livewire/admin/payment-table.blade.php

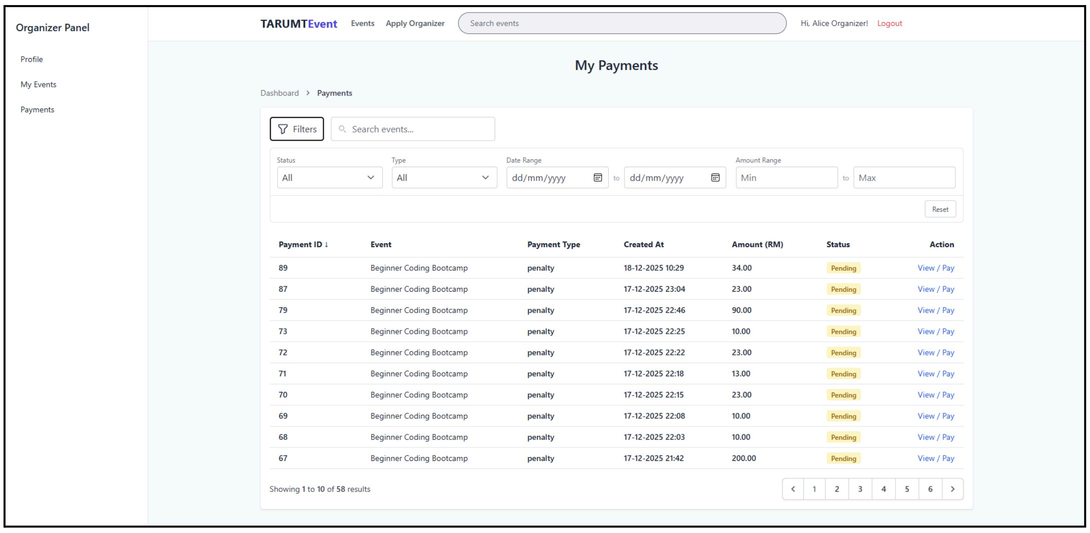


Payment ID ↓	Event	Payment Type	Created At	Amount (RM)	Status	Action
89	Beginner Coding Bootcamp (Alice Organizer)	penalty	18-12-2025 10:29	34.00	Pending	View
87	Beginner Coding Bootcamp (Alice Organizer)	penalty	17-12-2025 23:04	23.00	Pending	View
79	Beginner Coding Bootcamp (Alice Organizer)	penalty	17-12-2025 22:46	90.00	Pending	View
73	Beginner Coding Bootcamp (Alice Organizer)	penalty	17-12-2025 22:25	10.00	Pending	View
72	Beginner Coding Bootcamp (Alice Organizer)	penalty	17-12-2025 22:22	23.00	Pending	View
71	Beginner Coding Bootcamp (Alice Organizer)	penalty	17-12-2025 22:18	13.00	Pending	View
70	Beginner Coding Bootcamp (Alice Organizer)	penalty	17-12-2025 22:15	23.00	Pending	View
69	Beginner Coding Bootcamp (Alice Organizer)	penalty	17-12-2025 22:08	10.00	Pending	View
68	Beginner Coding Bootcamp (Alice Organizer)	penalty	17-12-2025 22:03	10.00	Pending	View
67	Beginner Coding Bootcamp (Alice Organizer)	penalty	17-12-2025 21:42	200.00	Pending	View

Showing 1 to 10 of 75 results

Figure 2.1.1 Admin-side Payment List

Class path: \app\Livewire\Organizer\OrganizerPaymentTable.php



Payment ID ↓	Event	Payment Type	Created At	Amount (RM)	Status	Action
89	Beginner Coding Bootcamp	penalty	18-12-2025 10:29	34.00	Pending	View / Pay
87	Beginner Coding Bootcamp	penalty	17-12-2025 23:04	23.00	Pending	View / Pay
79	Beginner Coding Bootcamp	penalty	17-12-2025 22:46	90.00	Pending	View / Pay
73	Beginner Coding Bootcamp	penalty	17-12-2025 22:25	10.00	Pending	View / Pay
72	Beginner Coding Bootcamp	penalty	17-12-2025 22:22	23.00	Pending	View / Pay
71	Beginner Coding Bootcamp	penalty	17-12-2025 22:18	13.00	Pending	View / Pay
70	Beginner Coding Bootcamp	penalty	17-12-2025 22:15	23.00	Pending	View / Pay
69	Beginner Coding Bootcamp	penalty	17-12-2025 22:08	10.00	Pending	View / Pay
68	Beginner Coding Bootcamp	penalty	17-12-2025 22:03	10.00	Pending	View / Pay
67	Beginner Coding Bootcamp	penalty	17-12-2025 21:42	200.00	Pending	View / Pay

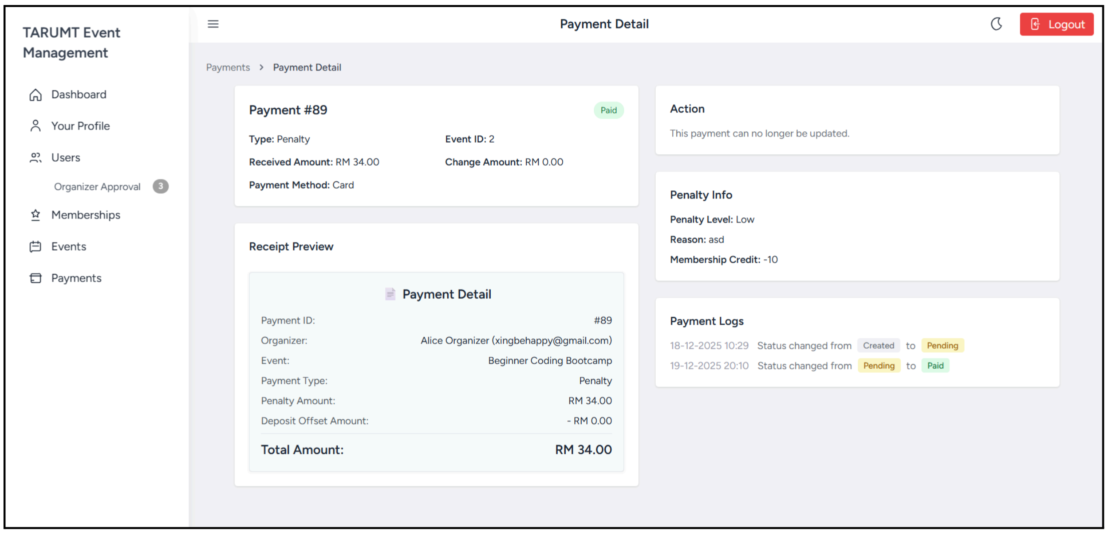
Showing 1 to 10 of 58 results

Figure 2.1.2 Organizer-side Payment List

2. View Payment Detail (Admin & Organizer)

The View Payment Details feature allows administrators and organizers to examine the specifics of a particular payment. The payment details interface displays key payment information, including payment type, status, amount, payment method and associated activity details such as related penalty, while also presenting a complete history of payment status changes. For administrators, the payment detail screen also serves as the entry point for permitted payment status updates when allowed by the system.

Class Path: \app\Http\Controller\PaymentController.php (function: paymentDetail)


TARUMT Event Management

Dashboard
Your Profile
Users
Organizer Approval 3
Memberships
Events
Payments

Payment Detail
Logout

Payments > Payment Detail

Payment #89 Paid
Type: Penalty
Event ID: 2
Received Amount: RM 34.00
Change Amount: RM 0.00
Payment Method: Card

Action
This payment can no longer be updated.

Penalty Info
Penalty Level: Low
Reason: asd
Membership Credit: -10

Payment Logs

18-12-2025 10:29	Status changed from	Created	to	Pending
19-12-2025 20:10	Status changed from	Pending	to	Paid

Receipt Preview

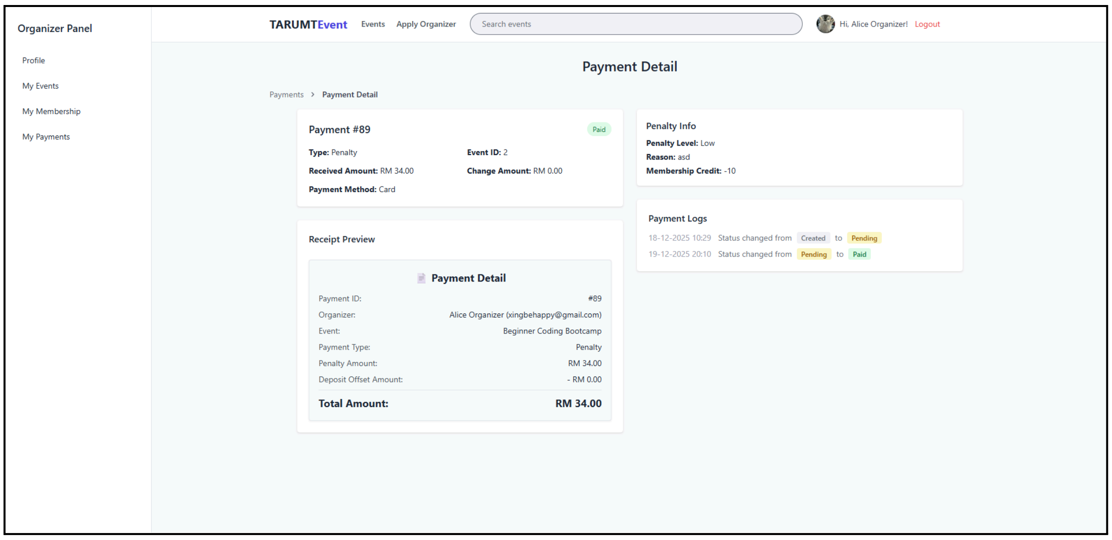
Payment Detail

Payment ID:	#89
Organizer:	Alice Organizer (xingbehappy@gmail.com)
Event:	Beginner Coding Bootcamp
Payment Type:	Penalty
Penalty Amount:	RM 34.00
Deposit Offset Amount:	- RM 0.00
Total Amount:	**RM 34.00**

Figure 2.2.1 Admin-side View Payment Detail

For event organizers, the payment details view operates in read-only mode and is restricted to payments associated with their own events. Organizers can review payment information, receipts and payment logs but lack the authority to modify payment statuses. This feature ensures transparency for organizers while maintaining administrators' effective control over financial management.

Class Path: \app\Http\Controller\PaymentController.php (function: organizerPaymentDetail)


Organizer Panel

Profile
My Events
My Membership
My Payments

TARUMTEvent Events Apply Organizer Search events Hi, Alice Organizer! Logout

Payment Detail

Payments > Payment Detail

Payment #89 Paid
Type: Penalty
Event ID: 2
Received Amount: RM 34.00
Change Amount: RM 0.00
Payment Method: Card

Penalty Info
Penalty Level: Low
Reason: asd
Membership Credit: -10

Payment Logs

18-12-2025 10:29	Status changed from	Created	to	Pending
19-12-2025 20:10	Status changed from	Pending	to	Paid

Receipt Preview

Payment Detail

Payment ID:	#89
Organizer:	Alice Organizer (xingbehappy@gmail.com)
Event:	Beginner Coding Bootcamp
Payment Type:	Penalty
Penalty Amount:	RM 34.00
Deposit Offset Amount:	- RM 0.00
Total Amount:	**RM 34.00**

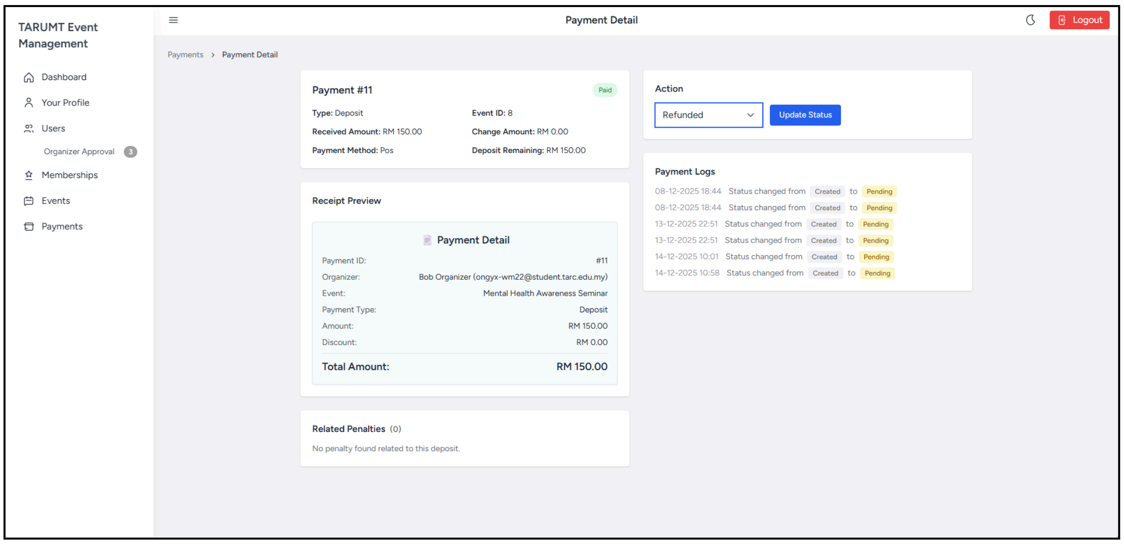
Figure 2.2.2: Organizer-side View Payment Detail

3. <u>Update Payment Status (Admin)</u>

The payment status update feature safeguards financial integrity by controlling changes to payment records. Only administrators can manually update payment statuses, including refund operations which are subject to strict eligibility rules to prevent premature or invalid refunds. Refunds are limited to eligible payments made within a specified period after the event concludes, ensuring all penalties and obligations are settled beforehand. If an event is canceled, the system automatically updates relevant payment statuses in the background without manual intervention.

This feature ensures payment status transitions remain consistently accurate, traceable and auditable.

Class Path: \app\Http\Controller\PaymentController.php (function: paymentDetail)



Payment #11 Paid

Type: Deposit
Event ID: 8
Received Amount: RM 150.00
Change Amount: RM 0.00
Payment Method: Pos
Deposit Remaining: RM 150.00

Action

Refunded

Payment Logs

08-12-2025 18:44 Status changed from Created to Pending
08-12-2025 18:44 Status changed from Created to Pending
13-12-2025 22:51 Status changed from Created to Pending
13-12-2025 22:51 Status changed from Created to Pending
14-12-2025 10:01 Status changed from Created to Pending
14-12-2025 10:58 Status changed from Created to Pending

Receipt Preview

Payment Detail

Payment ID:	#11
Organizer:	Bob Organizer (ongyx-wm22@student.tarc.edu.my)
Event:	Mental Health Awareness Seminar
Payment Type:	Deposit
Amount:	RM 150.00
Discount:	RM 0.00
Total Amount:	**RM 150.00**

Related Penalties (0)

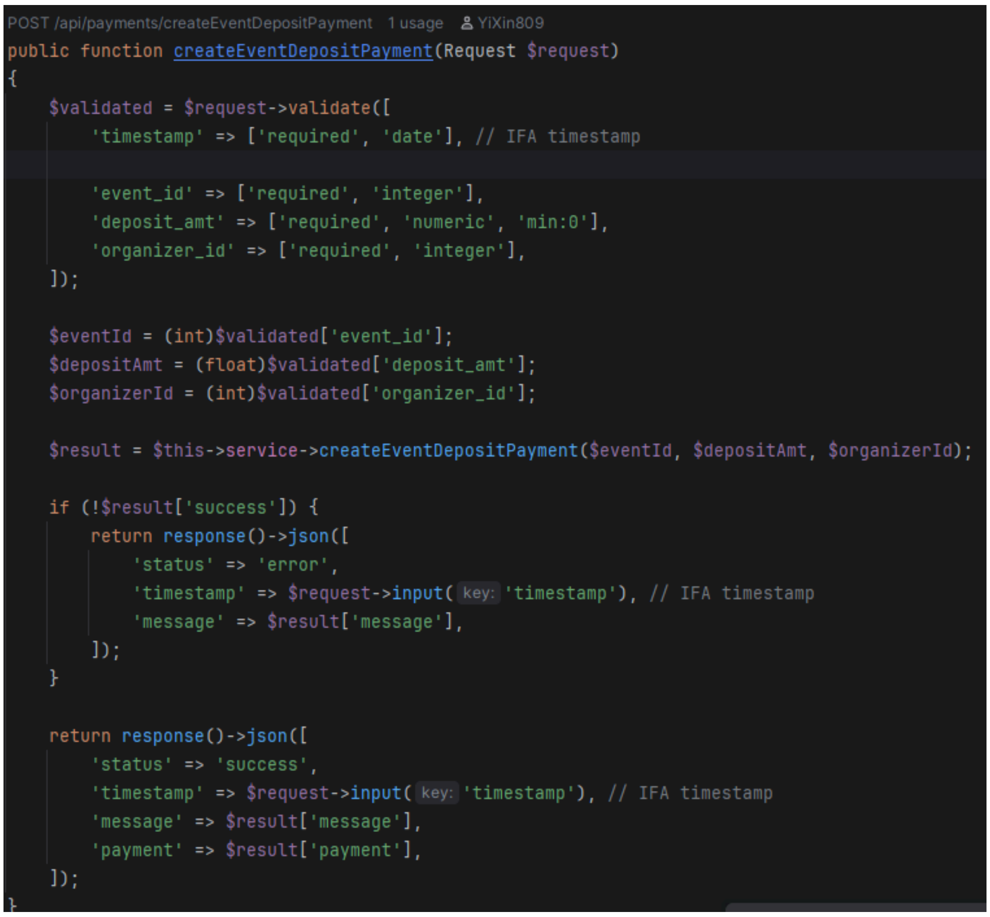
No penalty found related to this deposit.

Figure 2.3.1: Admin Update Payment Status

4. <u>Create Event Deposit Payment</u>

The Create Event Deposit Payment is designed to generate mandatory deposit payments after an event creation. This deposit serves as a financial guarantee to ensure organizers comply with event regulations. Upon event creation, the system automatically generates a deposit payment record linked to the event and organizer, containing key details such as deposit amount, payment purpose and status. Only valid deposit payments can be created and enter the payment process, ensuring financial consistency and rule enforcement. This feature assists administrators in managing event commitments while providing organizers with clear, traceable payment obligations.

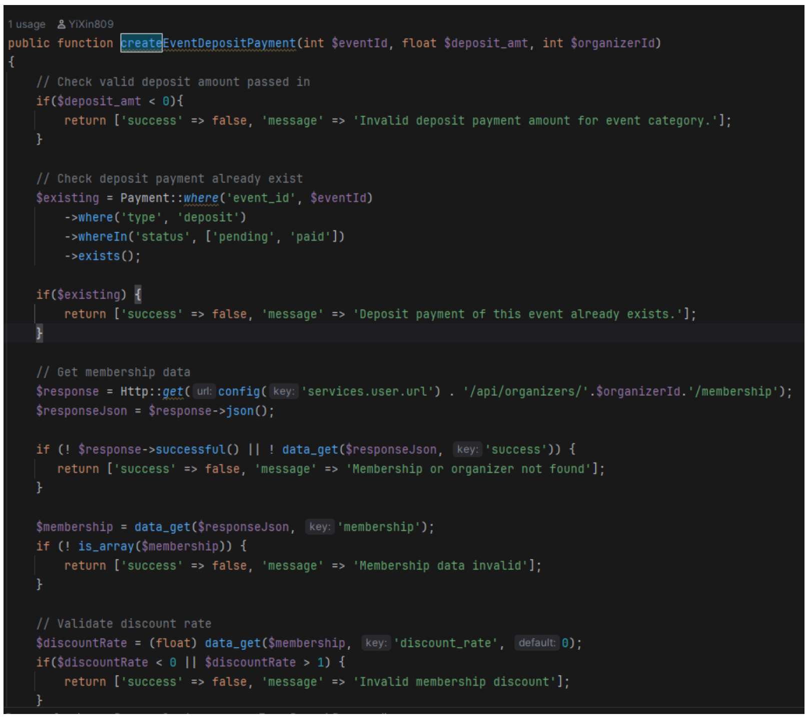
Class Path: \app\Http\Controller/PaymentController.php (function: createEventDepositPayment)



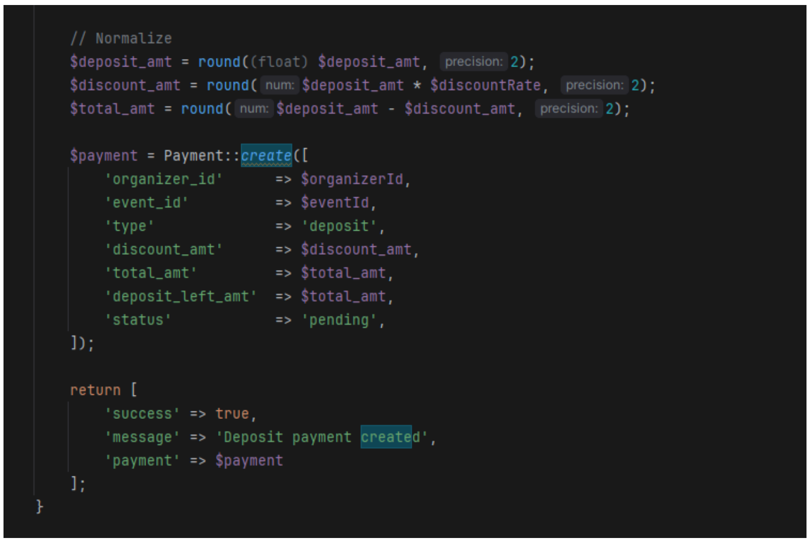
```
POST /api/payments/createEventDepositPayment   1 usage   YiXin809
public function createEventDepositPayment(Request $request)
{
    $validated = $request->validate([
        'timestamp' => ['required', 'date'], // IFA timestamp

        'event_id' => ['required', 'integer'],
        'deposit_amt' => ['required', 'numeric', 'min:0'],
        'organizer_id' => ['required', 'integer'],
    ]);

    $eventId = (int)$validated['event_id'];
    $depositAmt = (float)$validated['deposit_amt'];
    $organizerId = (int)$validated['organizer_id'];

    $result = $this->service->createEventDepositPayment($eventId, $depositAmt, $organizerId);

    if (!$result['success']) {
        return response()->json([
            'status' => 'error',
            'timestamp' => $request->input( key: 'timestamp'), // IFA timestamp
            'message' => $result['message'],
        ]);
    }

    return response()->json([
        'status' => 'success',
        'timestamp' => $request->input( key: 'timestamp'), // IFA timestamp
        'message' => $result['message'],
        'payment' => $result['payment'],
    ]);
}
```

Figure 2.4.1 Code Snippet of Create Event Deposit Controller

Class Path: \app\Services\PaymentService.php (function: createEventDepositPayment)



```
1 usage   YiXin809
public function createEventDepositPayment(int $eventId, float $deposit_amt, int $organizerId)
{
    // Check valid deposit amount passed in
    if($deposit_amt < 0){
        return ['success' => false, 'message' => 'Invalid deposit payment amount for event category.'];
    }

    // Check deposit payment already exist
    $existing = Payment::where('event_id', $eventId)
        ->where('type', 'deposit')
        ->whereIn('status', ['pending', 'paid'])
        ->exists();

    if($existing) {
        return ['success' => false, 'message' => 'Deposit payment of this event already exists.'];
    }

    // Get membership data
    $response = Http::get( url: config( key: 'services.user.url') . '/api/organizers/'.$organizerId.'/membership');
    $responseJson = $response->json();

    if (! $response->successful() || ! data_get($responseJson,  key: 'success')) {
       return ['success' => false, 'message' => 'Membership or organizer not found'];
    }

    $membership = data_get($responseJson,  key: 'membership');
    if (! is_array($membership)) {
        return ['success' => false, 'message' => 'Membership data invalid'];
    }

    // Validate discount rate
    $discountRate = (float) data_get($membership,  key: 'discount_rate',  default: 0);
    if($discountRate < 0 || $discountRate > 1) {
        return ['success' => false, 'message' => 'Invalid membership discount'];
    }
```



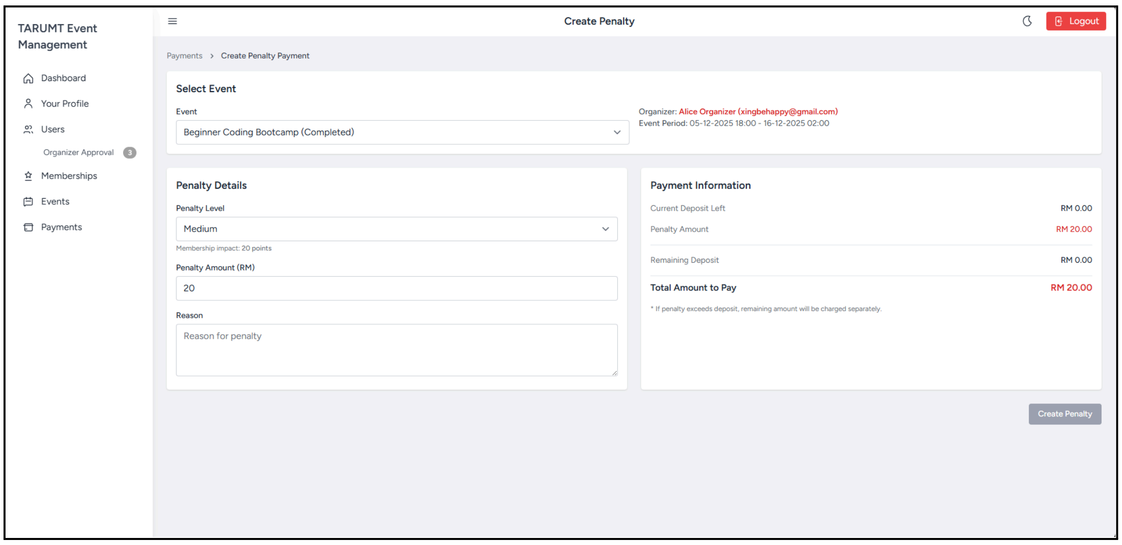
```
    // Normalize
    $deposit_amt = round((float) $deposit_amt,  precision: 2);
    $discount_amt = round( num: $deposit_amt * $discountRate,  precision: 2);
    $total_amt = round( num: $deposit_amt - $discount_amt,  precision: 2);

    $payment = Payment::create([
        'organizer_id'      => $organizerId,
        'event_id'          => $eventId,
        'type'              => 'deposit',
        'discount_amt'      => $discount_amt,
        'total_amt'         => $total_amt,
        'deposit_left_amt'  => $total_amt,
        'status'            => 'pending',
    ]);

    return [
        'success' => true,
        'message' => 'Deposit payment created',
        'payment' => $payment
    ];
}
```

Figure 2.4.2: Code Snippet of Create Event Deposit Service

5. <u>Create Penalty Payment (Admin)</u>

The penalty payment feature was created to impose financial penalties on organizers who violate event rules or fail to fulfill necessary obligations. This feature applies only to eligible events to ensure penalties are enforced within a manageable timeframe. When a penalty is created, the system automatically deducts the penalty amount from the event's remaining deposit. If the deposit sufficiently covers the penalty, the system immediately marks it as paid without requiring additional action from the organizer. When the deposit is insufficient, the remaining amount will be recorded as an outstanding balance. Additionally, the system will deduct member credit score based on the selected penalty level, ensuring financial penalties align with membership consequences. This feature achieves fairness in rule enforcement while safeguarding financial accuracy and accountability.

Class Path:\app\Livewire\Admin\Penalty\CreatePenaltyPayment.php



Select Event

Event: Beginner Coding Bootcamp (Completed)

Organizer: Alice Organizer (xingbehappy@gmail.com)
Event Period: 05-12-2025 18:00 - 16-12-2025 02:00

Penalty Details

Penalty Level: Medium
Membership impact: 20 points

Penalty Amount (RM): 20

Reason: Reason for penalty

Payment Information

Current Deposit Left	RM 0.00
Penalty Amount	RM 20.00
Remaining Deposit	RM 0.00
Total Amount to Pay	RM 20.00

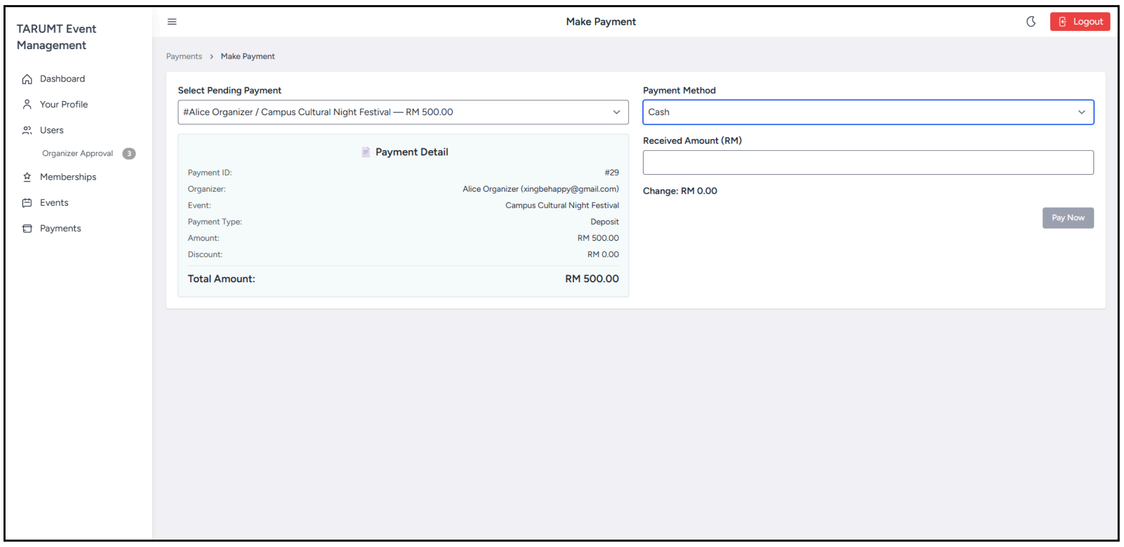
* If penalty exceeds deposit, remaining amount will be charged separately.

Figure 2.5.1: System Screen of Create Penalty Event

6. <u>Make Payment (Admin)</u>

The Administrator Payment feature enables administrators to manually record payments received through offline methods such as cash or point-of-sale systems. This functionality is restricted to administrators to ensure controlled processing of manual payments. Only payments in a pending status can be processed via this feature. When making a payment, administrators must select the payment method and record the amount received if paying in cash. The system verifies the received amount for sufficiency, automatically calculates any change due and marks the payment status as Paid. This feature ensures accurate recording of offline payments while maintaining financial data consistency and auditability.

Class Path:\app\Livewire\Admin\MakePayment.php



Select Pending Payment: #Alice Organizer / Campus Cultural Night Festival — RM 500.00

Payment Detail

Payment ID:	#29
Organizer:	Alice Organizer (xingbehappy@gmail.com)
Event:	Campus Cultural Night Festival
Payment Type:	Deposit
Amount:	RM 500.00
Discount:	RM 0.00
Total Amount:	**RM 500.00**

Payment Method: Cash

Received Amount (RM):

Change: RM 0.00

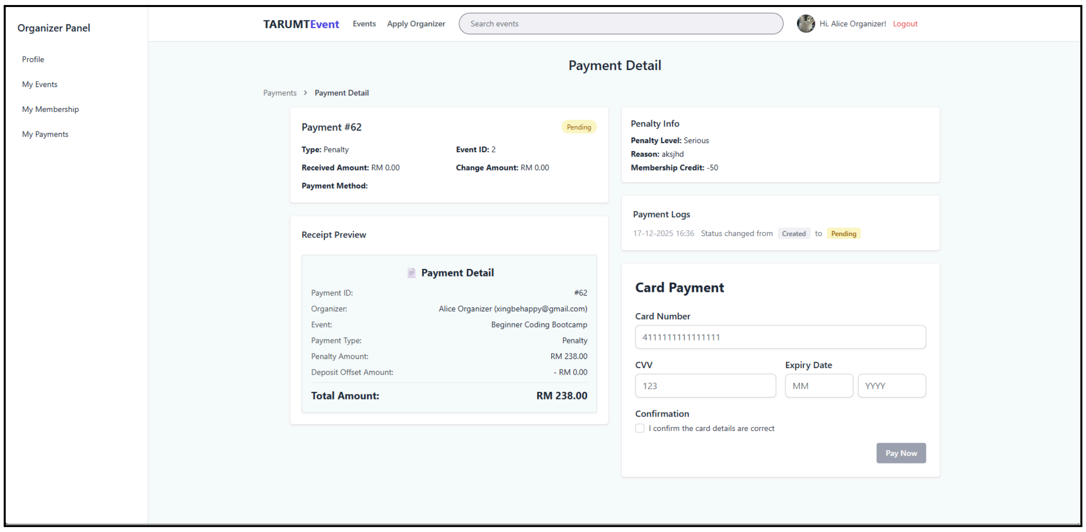
Figure 2.6.1: System Screen for Admin Record Payment

7. Make Payment (Organizer)

The organizer payment feature enables organizers to settle outstanding balances directly within the system via online card payments. Organizers can only view and pay for their own events, ensuring ownership and access control. This feature applies only to pending payments and supports two payment types which are deposits and penalties. Upon successful payment, the system automatically records the payment method, updates the status to “Paid” and retains the payment timestamp. This functionality enables organizers to efficiently fulfill financial obligations while maintaining secure and accurate payment records.

There is a card payment form provided in the payment detail of the unpaid payments. Organizers can directly fill in their card details and pay.

Class Path:\app\Livewire\Organizer\MakePayment.php


Organizer Panel
Profile
My Events
My Membership
My Payments

TARUMTEvent Events Apply Organizer Search events Hi, Alice Organizer! Logout

Payment Detail

Payments > Payment Detail

Payment #62 Pending
Type: Penalty **Event ID:** 2
Received Amount: RM 0.00 **Change Amount:** RM 0.00
Payment Method:

Penalty Info
Penalty Level: Serious
Reason: aksjhd
Membership Credit: -50

Payment Logs
17-12-2025 16:36 Status changed from Created to Pending

Receipt Preview

Payment Detail

Payment ID:	#62
Organizer:	Alice Organizer (xingbehappy@gmail.com)
Event:	Beginner Coding Bootcamp
Payment Type:	Penalty
Penalty Amount:	RM 238.00
Deposit Offset Amount:	- RM 0.00
Total Amount:	**RM 238.00**

Card Payment
Card Number: 4111111111111111
CVV: 123
Expiry Date: MM YYYY
Confirmation
I confirm the card details are correct
Pay Now

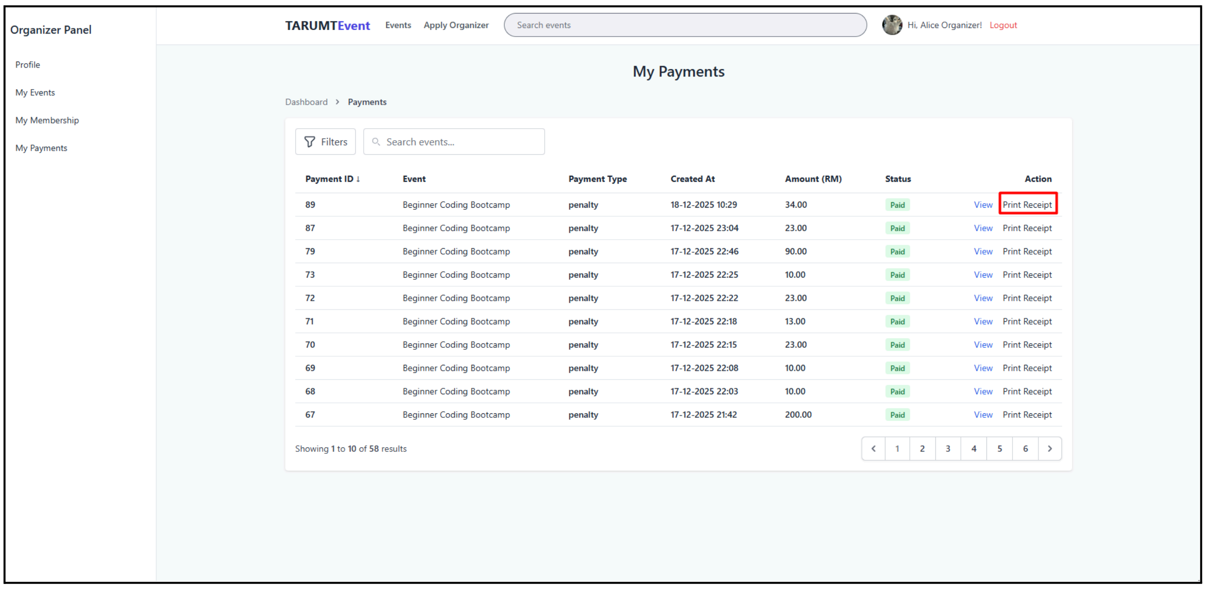
Figure 2.7.1: Card Payment Form for Organizer Make Payment

8. Print Receipt

The receipt printing feature enables administrators and organizers to generate and view official payment receipts for completed transactions. Receipts are only applicable to transactions with a paid status, ensuring only finalized transactions are recorded. Organizers can only view receipts for payments they made, while administrators have access to all payment receipts. When requesting a receipt, the system retrieves relevant activity information and generates a PDF receipt for preview or download. This feature provides administrators and organizers with clear financial documentation, supporting transparent record management.

Admin or organizer can direct to the Payment List page, the payment which is available to print receipt will have a "Print Receipt" button at the “Action” column.

Class Path: \app\Livewire\Organizer\OrganizerPaymentTable.php


Organizer Panel
Profile
My Events
My Membership
My Payments

TARUMTEvent Events Apply Organizer Search events Hi, Alice Organizer! Logout

My Payments

Dashboard > Payments

Filters Search events...

Payment ID ↓	Event	Payment Type	Created At	Amount (RM)	Status	Action
89	Beginner Coding Bootcamp	penalty	18-12-2025 10:29	34.00	Paid	View Print Receipt
87	Beginner Coding Bootcamp	penalty	17-12-2025 23:04	23.00	Paid	View Print Receipt
79	Beginner Coding Bootcamp	penalty	17-12-2025 22:46	90.00	Paid	View Print Receipt
73	Beginner Coding Bootcamp	penalty	17-12-2025 22:25	10.00	Paid	View Print Receipt
72	Beginner Coding Bootcamp	penalty	17-12-2025 22:22	23.00	Paid	View Print Receipt
71	Beginner Coding Bootcamp	penalty	17-12-2025 22:18	13.00	Paid	View Print Receipt
70	Beginner Coding Bootcamp	penalty	17-12-2025 22:15	23.00	Paid	View Print Receipt
69	Beginner Coding Bootcamp	penalty	17-12-2025 22:08	10.00	Paid	View Print Receipt
68	Beginner Coding Bootcamp	penalty	17-12-2025 22:03	10.00	Paid	View Print Receipt
67	Beginner Coding Bootcamp	penalty	17-12-2025 21:42	200.00	Paid	View Print Receipt

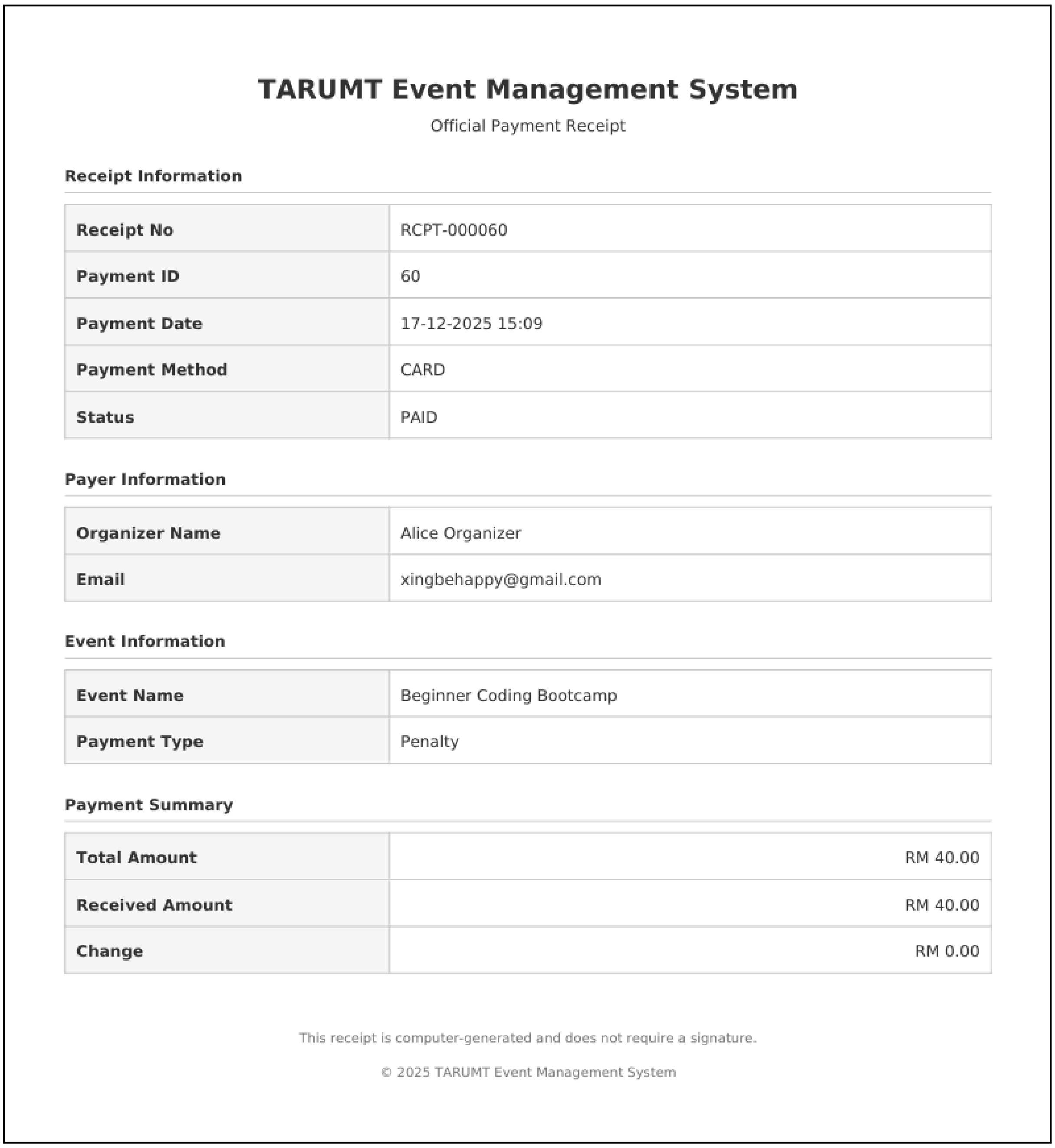
Showing 1 to 10 of 58 results

< 1 2 3 4 5 6 >

Figure 2.8.1: Print Receipt Button on Organizer or Admin Payment List

Then will open a new tab and display the preview of the PDF receipt.

Class Path: \app\Http\Controller\PaymentControlelr.php (function: receipt)



TARUMT Event Management System

Official Payment Receipt

Receipt Information

Receipt No	RCPT-000060
Payment ID	60
Payment Date	17-12-2025 15:09
Payment Method	CARD
Status	PAID

Payer Information

Organizer Name	Alice Organizer
Email	xingbehappy@gmail.com

Event Information

Event Name	Beginner Coding Bootcamp
Payment Type	Penalty

Payment Summary

Total Amount	RM 40.00
Received Amount	RM 40.00
Change	RM 0.00

This receipt is computer-generated and does not require a signature.

© 2025 TARUMT Event Management System

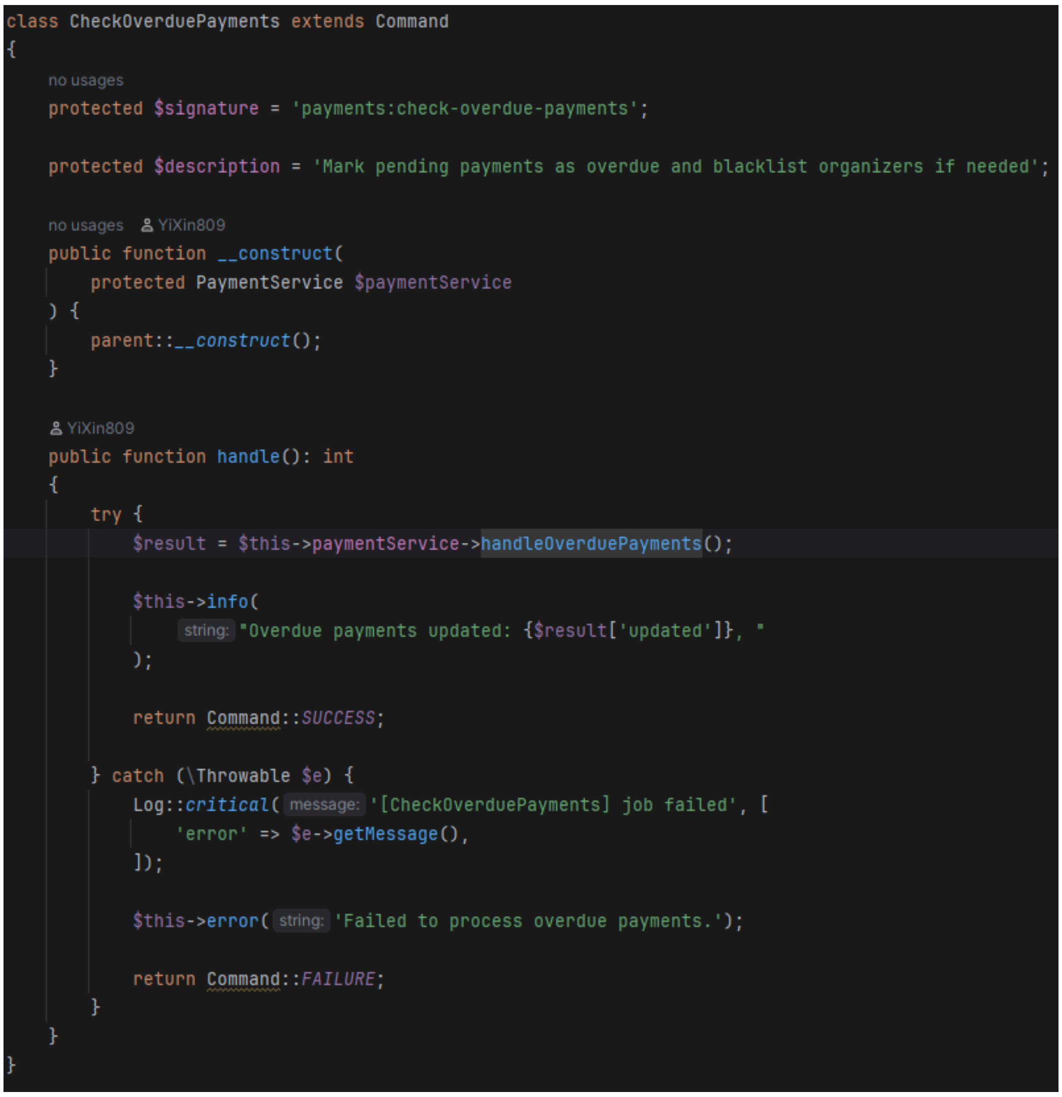
Figure 2.8.2: Preview PDF Receipt

9. **<u>Check Overdue Payments (Scheduler)</u>**

The Check Overdue Payments scheduler is designed to automatically enforce deposit payment deadlines without manual intervention. The system scans all pending deposit payments at preset intervals, identifies payments remaining unpaid beyond a 14-day grace period and automatically flags such payments as overdue.

Upon detecting an overdue deposit payment, the system simultaneously cancels related events to prevent their progression without valid deposit receipt. Each affected event undergoes processing only once throughout the entire workflow, ensuring consistency and controllability in execution. This automated mechanism safeguards payment compliance by preventing invalid or unpaid activities from remaining active in the system and reduces the need for manual administrative operations.

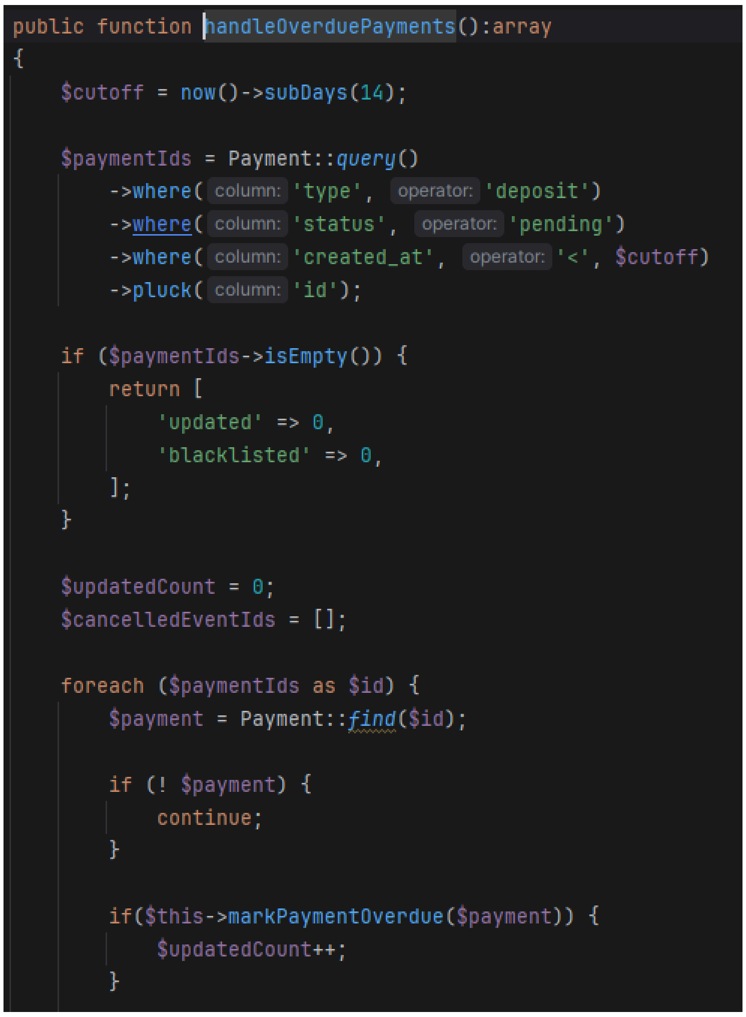
Class Path: \app\Console\Command\CheckOverduePayment.php



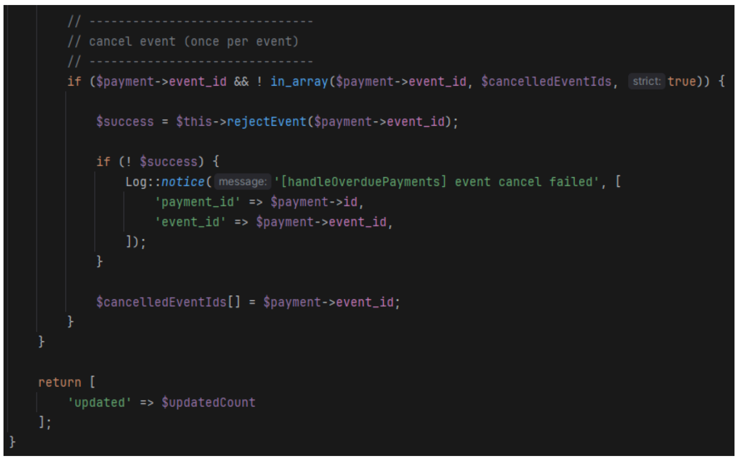
```php
class CheckOverduePayments extends Command
{
    no usages
    protected $signature = 'payments:check-overdue-payments';

    protected $description = 'Mark pending payments as overdue and blacklist organizers if needed';

    no usages  YiXin809
    public function __construct(
        protected PaymentService $paymentService
    ) {
        parent::__construct();
    }

    YiXin809
    public function handle(): int
    {
        try {
            $result = $this->paymentService->handleOverduePayments();

            $this->info(
                string: "Overdue payments updated: {$result['updated']}, "
            );

            return Command::SUCCESS;

        } catch (\Throwable $e) {
            Log::critical( message: '[CheckOverduePayments] job failed', [
                'error' => $e->getMessage(),
            ]);

            $this->error( string: 'Failed to process overdue payments.');

            return Command::FAILURE;
        }
    }
}
```

Figure 2.9.1: Code Snippet of Check Overdue Payment Command

Class Path: \app\Services\PaymentServices.php (function: handleOverduePayments)



```php
public function handleOverduePayments():array
{
    $cutoff = now()->subDays(14);

    $paymentIds = Payment::query()
        ->where( column: 'type',  operator: 'deposit')
        ->where( column: 'status',  operator: 'pending')
        ->where( column: 'created_at',  operator: '<', $cutoff)
        ->pluck( column: 'id');

    if ($paymentIds->isEmpty()) {
        return [
            'updated' => 0,
            'blacklisted' => 0,
        ];
    }

    $updatedCount = 0;
    $cancelledEventIds = [];

    foreach ($paymentIds as $id) {
        $payment = Payment::find($id);

        if (! $payment) {
            continue;
        }

        if($this->markPaymentOverdue($payment)) {
            $updatedCount++;
        }
```



```php
        // -----------------------------
        // cancel event (once per event)
        // -----------------------------
        if ($payment->event_id && ! in_array($payment->event_id, $cancelledEventIds,  strict: true)) {

            $success = $this->rejectEvent($payment->event_id);

            if (! $success) {
                Log::notice( message: '[handleOverduePayments] event cancel failed', [
                    'payment_id' => $payment->id,
                    'event_id' => $payment->event_id,
                ]);
            }

            $cancelledEventIds[] = $payment->event_id;
        }
    }

    return [
        'updated' => $updatedCount
    ];
}
```

Figure 2.9.2: Code Snippet of Handle Overdue Payment Service

3. Entity Classes

Payment Entity

The Payment entity represents financial transactions within the system, encompassing event deposits and penalty fees. It stores all critical payment information, including amounts due, discounts, payment methods, transaction status and payment timestamps.

As the core entity of the payment module, Payment serves as the centralized record of transaction financial status. Business rules and payment processes are handled externally to maintain separation of duties.

Relationship:

- Organizer: Each payment is associated with an organizer, representing the user responsible for the payment.
- Event: Payments can be linked to events, supporting event-based payment tracking and verification.
- PaymentLog: A single payment can be associated with multiple payment logs to record status changes and support audit trails.
- PaymentPenaltyDetail: Payments can optionally be configured with individual penalty details to store specific penalty information, enabling payment entities to reuse processing for different payment types.

PaymentLog Entity

The PaymentLog entity records the historical trajectory of payment transaction status changes. It stores past records and updates information of payment statuses, enabling the system to track the progression of payments through various states across different time periods。

By preserving state transition histories without embedding historical data within the payment entity itself, PaymentLog achieves auditability and traceability. This design allows payment entities to focus on their current financial state while reliably documenting past changes.

Relationships:

- Payment: Each payment log is associated with a single payment transaction, representing the recorded state changes for that transaction. A single payment transaction may be linked to multiple payment logs, forming a complete audit trail of payment state transitions.

PaymentPenaltyDetail Entity

The PaymentPenaltyDetail entity represents details related to penalty payments. It stores penalty-specific data such as penalty level, reason, amount of deposit deducted and membership point deductions associated with the penalty transaction.

PaymentPenaltyDetail complements the Payment entity by encapsulating penalty-specific attributes separately. This enables the Payment entity to remain reusable across different payment types while maintaining clear separation of responsibilities.

Relationships:

- Payment: Each PaymentPenaltyDetail entity is associated with a single payment transaction. A payment record optionally contains one penalty detail record, depending on whether the transaction is a penalty-type payment.

Entity Class Diagram

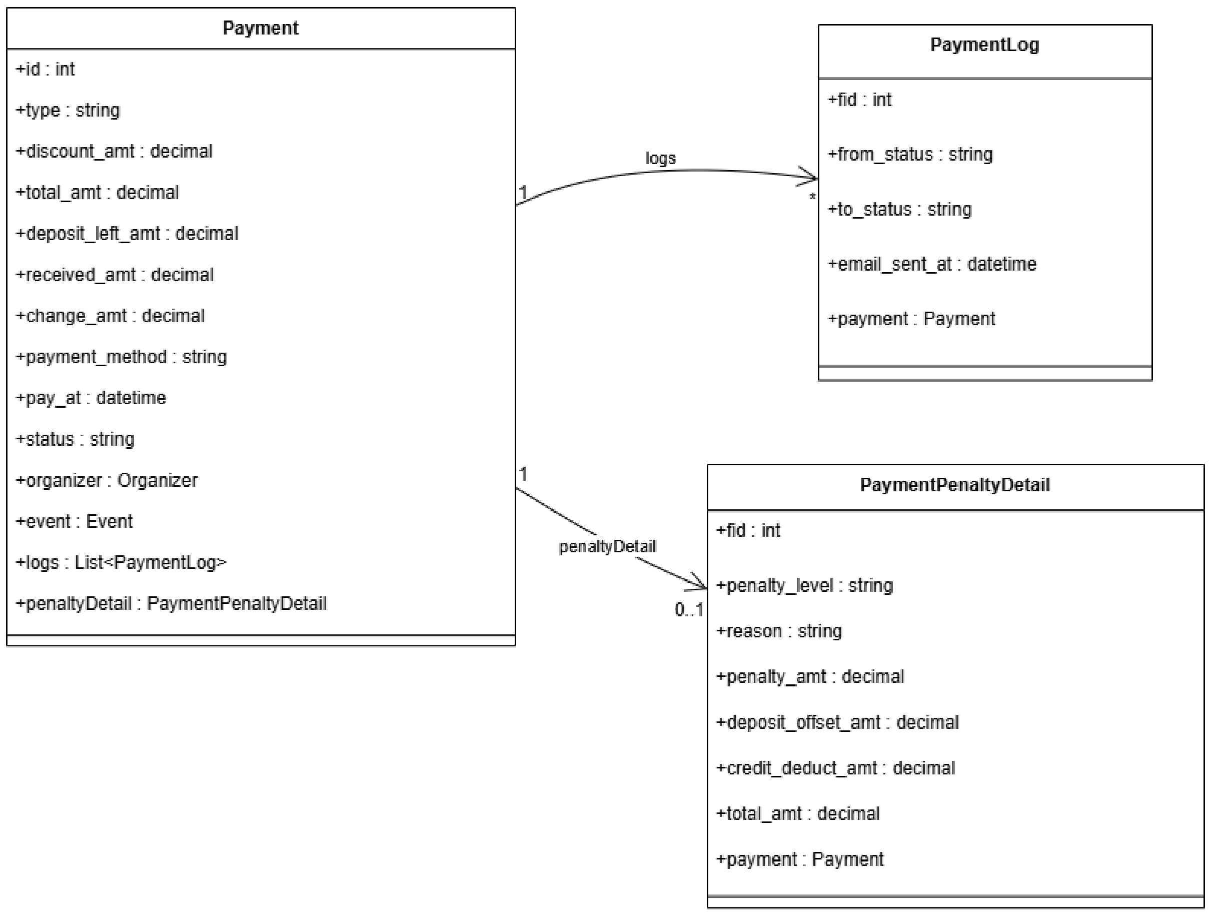


Figure 3.1: Entity Class Diagram of Payment Module

4. Design Pattern

4.1 Description of Design Pattern

Purpose
The Observer pattern is implemented within the payment module to monitor payment lifecycle events, specifically payment creation and payment status changes. Its purpose is to ensure that all payment-related side effects such as payment log recording, notification triggering, receipt preparation and compliance checks are consistently executed whenever a payment status changes. By centralizing the detection of payment changes at the model layer, the system guarantees that no critical payment events are overlooked, regardless of how payments are created or updated. It also prevents inconsistent execution across different payment workflows.

How does it work?

1. **PaymentObserver**

   PaymentObserver registers to monitor the payment model and serves as the primary trigger point for the observer pattern. It listens for model lifecycle events and translates these into meaningful domain events.

   Upon creating a new payment record, the observer immediately creates a payment log entry to record the initial state, ensuring every payment has an audit trail from creation. After logging, the observer distributes a PaymentStatusChanged event containing the new status.

   When payment is updated, the observer checks if the payment status has changed via `wasChanged('status')`. If the status remains unchanged, it exits early to avoid redundant processing. Upon detecting a status change, the observer retrieves the previous status, logs a new payment log entry, and simultaneously distributes a PaymentStatusChanged event carrying both the old and new statuses.

   This design ensures all payment status changes are captured at the model layer, regardless of whether they originate from administrator actions, organizer payments, automated schedulers, or API calls.

2. **PaymentStatusChanged Event**

   The payment status change event serves as the communication medium between the observer and all relevant listeners, encapsulating all necessary contextual information required for payment state transitions, including payment model, previous state, new state and relevant payment log identifier.

   By encapsulating this data into a single event, the system avoids redundant database queries while ensuring each listener receives a consistent and complete snapshot of the payment change. This event-driven pattern enables multiple independent listeners to respond to the same payment status change without requiring direct dependencies between them.

3. **SendPaymentStatusEmail Listener**

   The SendPaymentStatusEmail listener handles notification emails for payment status changes excluding paid status, which is handled separately due to different email body content and attachment. Its handle() method performs the following responsibilities:

   - Status filtering

     The listener exits early if the new status is paid, ensuring that only non-paid transitions such as cancelled, overdue or refunded are processed here.

   - Idempotency control

     It retrieves the related PaymentLog and checks whether an email has already been sent. This prevents duplicate emails in retry or failure scenarios.

   - Cross-module data retrieval

     The listener fetches event and organizer information from the Event Module via web service calls, ensuring the Payment Module does not directly depend on Event Module data models.

   - Email dispatch

     A status-specific email is sent to the organizer using PaymentStatusMail, with different templates applied depending on whether the payment is a penalty or a deposit.

   - Post-processing update

     After successful delivery, the listener updates the payment log to mark the email as sent, ensuring reliable delivery tracking.

   This listener is executed asynchronously using queued jobs to avoid delaying the main payment transaction.

4. **SendPaymentReceiptEmail Listener**

   The SendPaymentReceiptEmail listener is dedicated to handling the paid status. Its handle() method enforces the following workflow:

   - Paid Status Validation

     If the new status is not paid, the listener immediately returns, ensuring strict separation of duties.

   - Event and Organizer Resolution

     Retrieves event details and organizer contact information via the event module API.

- Receipt Generation

  Generates the payment receipt PDF using the PaymentReceiptService. This operation is isolated from payment logic and triggers only after successful payment.

- Receipt Email Dispatch

  Sends the generated receipt as an attachment to the organizer via PaymentPaidMail.

- Cleanup and Security Checks

  Temporary PDF files are deleted immediately after use, and payment logs are updated to prevent duplicate receipt emails.

By encapsulating receipt generation as a dedicated listener, the system ensures the reliability and maintainability of payment confirmation and document delivery.

5. **RefreshOrganizerBlacklistStatus Listener**

This listener handles compliance enforcement for organizers related to penalty payments. Its responsibilities include:

- Payment Type Validation

  Respond only to penalty payments, ignoring other payment types to avoid unnecessary processing.

- Trigger Condition Evaluation

  Respond to penalty creation and status change events, ensuring blacklist status is recalculated when penalty obligations change.

- Blacklist Status Recalculation

  The listener tracks the number of pending penalty payments for an organizer. If this count exceeds a configured threshold, the organizer is marked as blacklisted.

- Cross-Module Updates

  Updates to blacklist status are sent to user modules via web service calls, enabling loosely coupled centralized user management.

This listener ensures organizer penalties directly impact their permissions while preventing blacklist logic from being embedded within payment services.

4.2 Implementation of Design Pattern

<u>Observer Class Diagram</u>

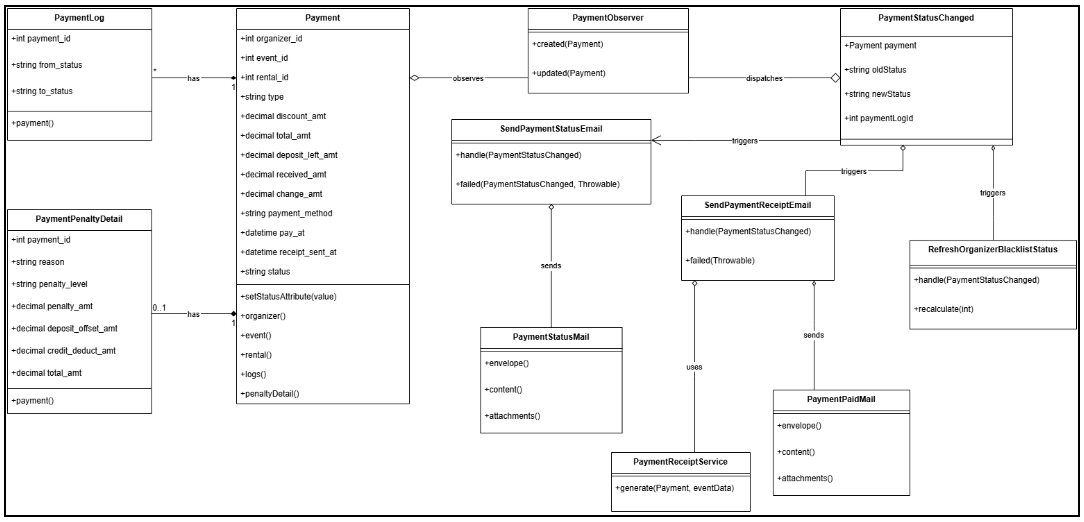


Figure 4.2: Class Diagram for Payment Module Observer Design Pattern

<u>Why is the Observer Suitable?</u>

1. **Payment State Changes Are Central Domain Events**

   Within the payment module, changes in payment status represent core domain events that directly influence system behavior. Payment creation and state transitions such as pending, paid, refunded and cancelled are not isolated operations but events that trigger multiple subsequent processes. The Observer pattern is particularly well-suited for this scenario, as it enables the system to automatically respond to these domain events without requiring explicit coordination from the payment controller or service.

2. **Multiple Independent Side Effects per Payment Change**

   A single payment status change may require executing multiple independent operations, including creating payment logs, sending email notifications, generating receipts and assessing organizer compliance. These operations are logically associated with the same payment event but do not depend on each other. The Observer pattern enables each responsibility to be handled by a separate listener, thereby preventing the core payment logic from becoming bloated or tightly coupled due to secondary processes.

3. **Multiple Entry Points for Payment Updates**

   Payment records can be created or updated through multiple system entry points, such as administrator actions, payments initiated by organizers, scheduled background tasks and internal API calls from other modules. Without an observer-based approach, each entry point would require manually triggering the same set of post-payment operations. By directly observing the payment model, the observer pattern ensures consistent behavior regardless of the source or location of payment changes.

4. **Support for Asynchronous and Long-Running Tasks**

Certain payment-related operations such as sending emails or generating PDF receipts consume significant resources and are unsuitable for synchronous execution within the request-response cycle. The observer pattern seamlessly integrates with queue listeners, enabling these operations to be processed asynchronously. This approach enhances system responsiveness while ensuring all necessary side effects are reliably executed after payment events occur.

5. **Decoupling of Payment Logic from External Modules**

   The payment module operates within a distributed system, interacting with the event module and user module via web services. Observer-based listeners centralize cross-module communication to respond to payment changes, rather than dispersing external API calls across controllers and services. This design reduces tight coupling between modules while simplifying maintenance when external service behaviors change.

6. **Extensibility for Future Business Rules**

   Payment-related business rules may change at any time such as new notification requirements, additional compliance checks or reporting features. The observer pattern achieves scalability by allowing new listeners to be introduced without modifying existing payment logic. This minimizes regression risks and ensures new requirements can be incrementally added as the system evolves.

5. Software Security

5.1 Potential Threat/Attack

Data Breaches Through Unauthorized API Access

The payment module manages highly sensitive financial information, including payment records, deposit balances, penalty details,and payment statuses. A potential threat to this module involves unauthorized access to payment data APIs where attackers attempt to steal financial information without legitimate authorization. Since these APIs return structured and actionable data, successful access could lead to the exposure of confidential transaction details and user-related financial information.

For example, attackers may use tools like Postman, cURL or automated API scanners to directly send HTTP requests to payment-related endpoints. Bypassing official frontend systems, attackers target backend URLs directly and attempt to guess or enumerate identifiers such as event IDs and organizer IDs. Through iterative modification of request parameters, attackers may obtain multiple payment records belonging to different events or organizers. The obtained information may include payment amounts, deposit balances, penalty fees and payment statuses. This data can be leveraged for financial profiling, targeted social engineering attacks or fraud schemes. For instance, upon discovering an organizer with unpaid penalties, attackers might impersonate administrators to deceive victims into making fraudulent payments.

Information Disclosure Through Improper Error Handling

The payment module contains complex business rules for managing payment creation, status transitions, refunds, penalties and interactions with external services. Potential attack vectors include deliberately triggering system errors to extract internal information from error responses. If error handling mechanisms are not strictly controlled, attackers may gain insight into backend logic, validation rules or system dependencies.

For example, attackers may use tools such as Postman, browser developer tools or fuzz testing scripts to submit malformed requests, invalid status values or incomplete payloads to payment endpoints. By observing variations in error responses, attackers can infer the payment module's data validation mechanisms, conditions triggering specific failures and how the system interacts with other services. If detailed exception information or stack traces are exposed, attackers can identify internal class names, database fields or service URLs. This information can be leveraged to refine subsequent attacks, pinpoint vulnerable endpoints, or more efficiently exploit logical flaws, thereby increasing the risk of further compromise to the payment system.

5.2 Secure Coding Practice

Communication Security for Payment Data APIs

The payment module incorporates built-in communication security mechanisms designed to safeguard sensitive financial data during transmission and prevent unauthorized external access to payment-related APIs. Since the endpoints exposed by this module return critical information such as payment records, deposit balances, penalty details and payment statuses, ensuring the security of communication channels is essential for protecting the confidentiality and integrity of data exchange between systems.

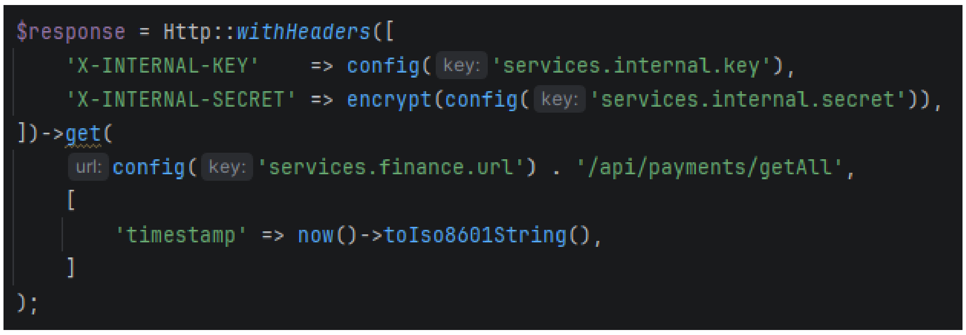
Within the payment module, all API interfaces used to retrieve or manipulate payment data enforce strict access requirements at the communication layer. Each request must include internal authentication credentials, which are securely configured via environment variables and never hardcoded in source code. Before transmission, the internal authentication key is encrypted and appended to the request header alongside the corresponding internal API key. Upon receiving a request, the payment module decrypts and validates these credentials before processing any payment-related operations. Requests lacking or containing invalid authentication information are immediately rejected. The module does not implicitly trust requests or responses from external services. Instead, it validates request parameters, confirms successful response statuses, and securely returns failure responses when communication with dependent services cannot be verified.

This measure directly mitigates the threat of unauthorized access to payment data APIs by preventing attackers from querying backend payment interfaces via direct HTTP requests. Even if attackers attempt to bypass the frontend and invoke the API using tools like Postman or cURL, requests lacking proper authentication context and valid request parameters will be rejected. Consequently, sensitive payment information remains consistently protected and accessible only to authorized system components, significantly reducing the risk of financial data breaches.

Code snippet:

- Client side call api with header

  Class Path: /app/Livewire/Admin/PaymentTable.php (function: list)

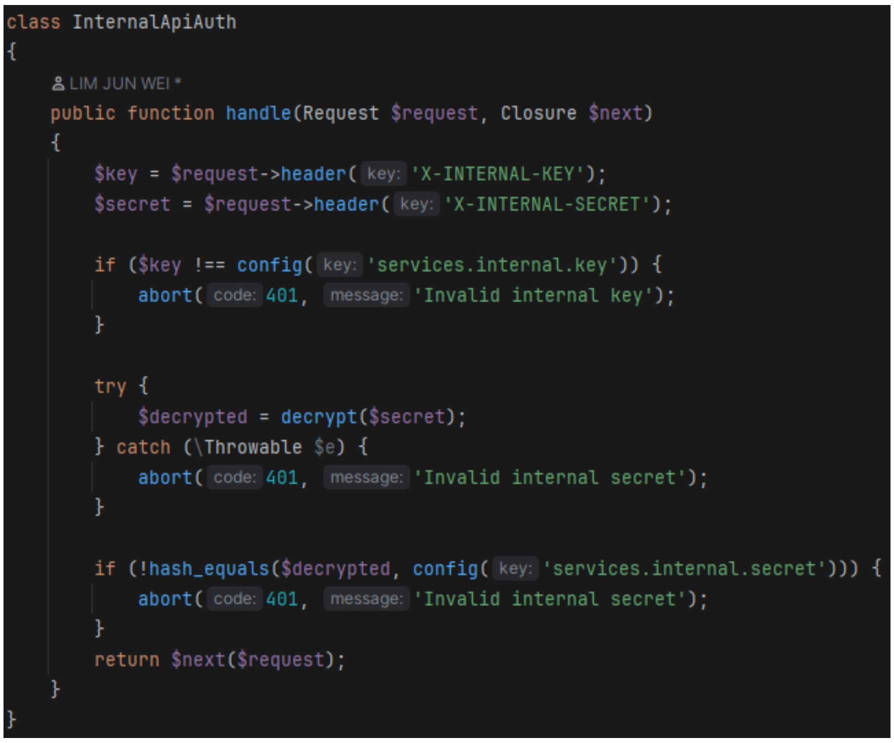


```php
$response = Http::withHeaders([
    'X-INTERNAL-KEY'    => config( key: 'services.internal.key'),
    'X-INTERNAL-SECRET' => encrypt(config( key: 'services.internal.secret')),
])->get(
    url: config( key: 'services.finance.url') . '/api/payments/getAll',
    [
        'timestamp' => now()->toIso8601String(),
    ]
);
```

Figure 5.2.1: Code Snippet of Calling API with Communication Security Handle

- Middleware to check key and secret before routing

  Class Path: \app\Http\Middleware\InternalApiAuth.php

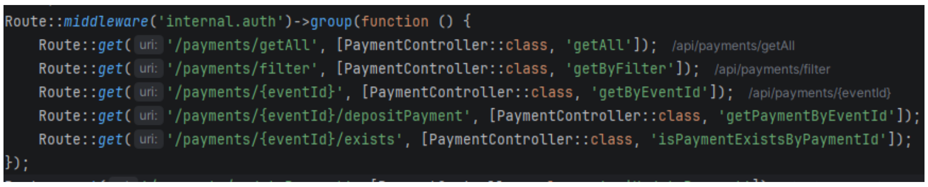


```
class InternalApiAuth
{
    LIM JUN WEI *
    public function handle(Request $request, Closure $next)
    {
        $key = $request->header( key: 'X-INTERNAL-KEY');
        $secret = $request->header( key: 'X-INTERNAL-SECRET');

        if ($key !== config( key: 'services.internal.key')) {
            abort( code: 401,  message: 'Invalid internal key');
        }

        try {
            $decrypted = decrypt($secret);
        } catch (\Throwable $e) {
            abort( code: 401,  message: 'Invalid internal secret');
        }

        if (!hash_equals($decrypted, config( key: 'services.internal.secret'))) {
            abort( code: 401,  message: 'Invalid internal secret');
        }
        return $next($request);
    }
}
```

Figure 5.2.2: Code Snippet of Checking HTTP Request Header

- Route wrap with InternalAPIAuth middleware

  Class Path: \routes\api.php



```
Route::middleware('internal.auth')->group(function () {
    Route::get( uri: '/payments/getAll', [PaymentController::class, 'getAll']);  /api/payments/getAll
    Route::get( uri: '/payments/filter', [PaymentController::class, 'getByFilter']);  /api/payments/filter
    Route::get( uri: '/payments/{eventId}', [PaymentController::class, 'getByEventId']);  /api/payments/{eventId}
    Route::get( uri: '/payments/{eventId}/depositPayment', [PaymentController::class, 'getPaymentByEventId']);
    Route::get( uri: '/payments/{eventId}/exists', [PaymentController::class, 'isPaymentExistsByPaymentId']);
});
```

Figure 5.2.3: Code Snippet of Route with InternalAuthApi

<u>**Controller Error Handling and Secure Logging**</u>

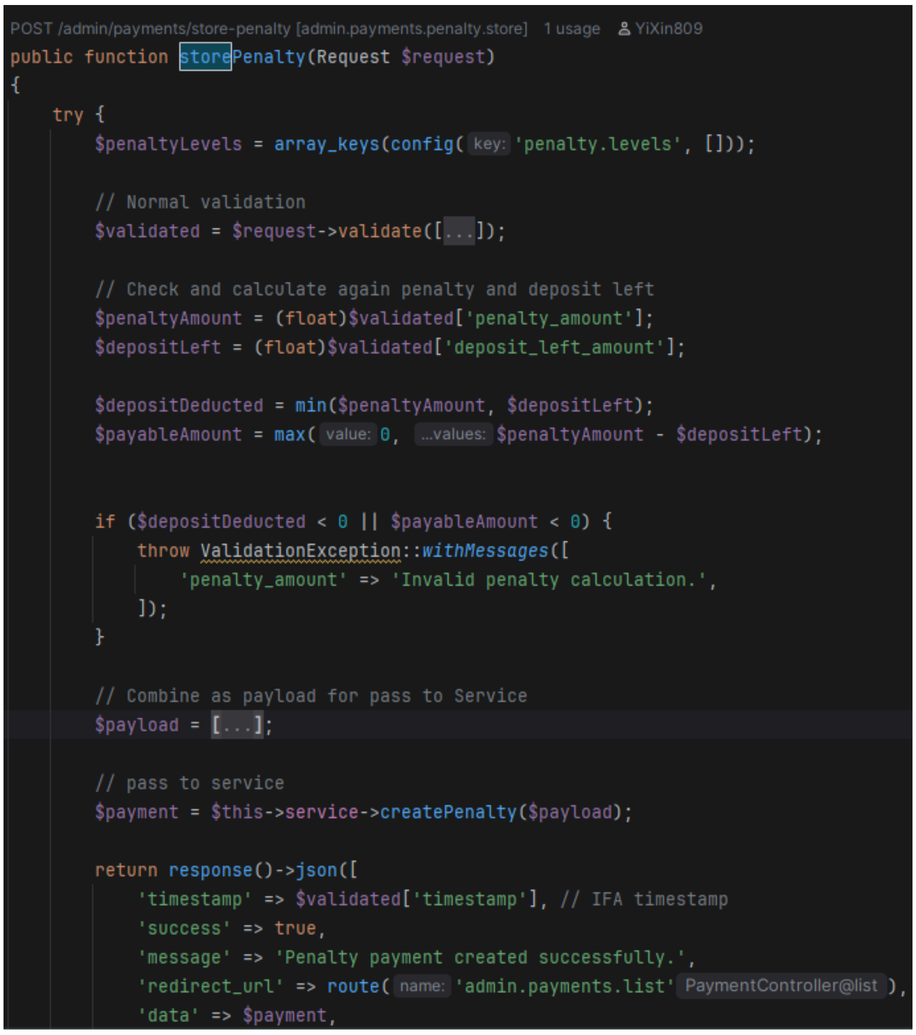
The payment module employs a controlled error handling strategy to ensure sensitive internal information is never exposed through client-facing responses, while maintaining traceability for critical backend operations. Due to the complexity of payment workflows such as penalty generation, deposit deductions, payment status transitions, and scheduled delinquency processing, runtime errors and business rule violations may occur and must be handled securely.

In the payment module, when an operation cannot be completed such as invalid payment status transitions or rule violations, the user-facing response always returns a generic failure message. For security-sensitive operations, the system selectively preserves detailed information, including exception messages and contextual data via server-side logging mechanisms. For example, failed operations during penalty generation, overdue payment processing and backend workflows are logged with associated identifiers to support auditing and incident investigations. Such logs reside within trusted server infrastructure and are not disclosed to end users.

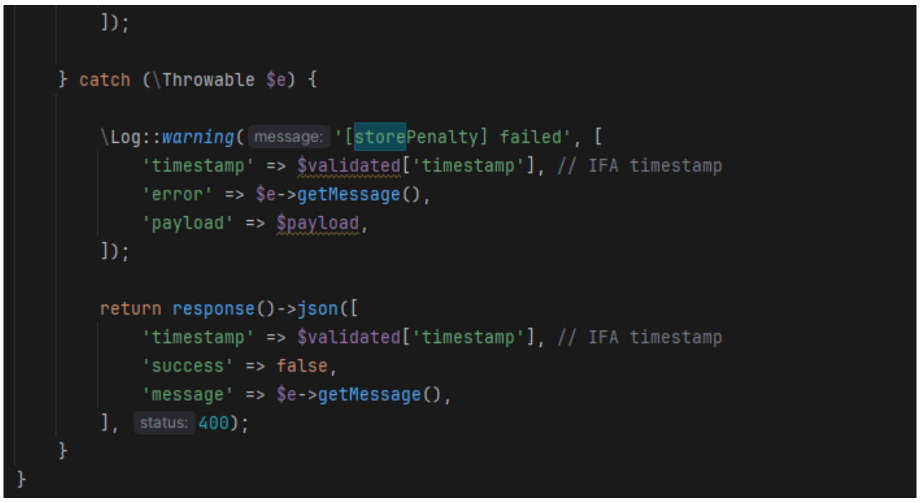
This approach mitigates information disclosure attacks by preventing attackers from inferring backend logic or system dependencies through erroneous responses. Even when attackers deliberately submit malformed requests or invalid status updates using tools like Postman or automated fuzz testing scripts, the system does not leak stack trace information, database structures or business rule details. Concurrently, server-side logging enables administrators to track abnormal behavior, analyze failed operations, and investigate suspicious activities without compromising system security.

Code snippet:

- Class path: \app\Http\Controller\PaymentControlelr.php (function: storePenalty)



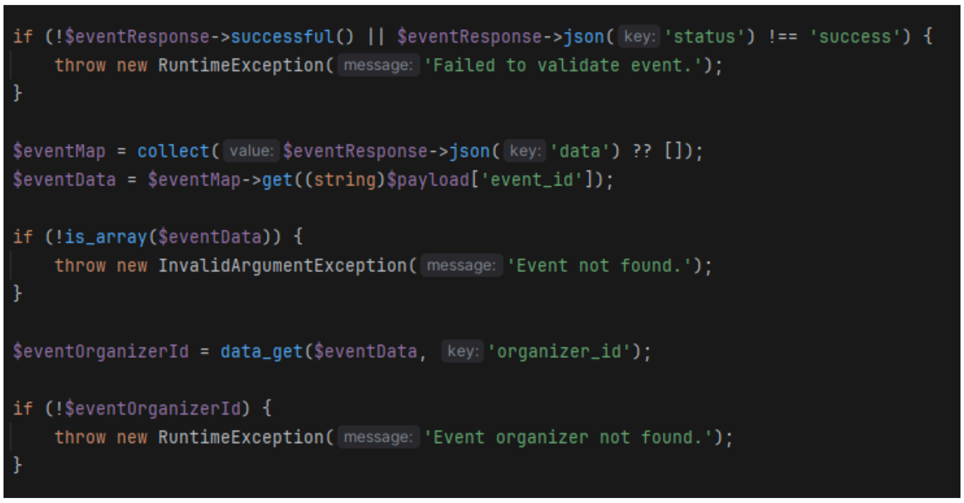
```
POST /admin/payments/store-penalty [admin.payments.penalty.store]   1 usage   YiXin809
public function storePenalty(Request $request)
{
    try {
        $penaltyLevels = array_keys(config( key: 'penalty.levels', []));

        // Normal validation
        $validated = $request->validate([...]);

        // Check and calculate again penalty and deposit left
        $penaltyAmount = (float)$validated['penalty_amount'];
        $depositLeft = (float)$validated['deposit_left_amount'];

        $depositDeducted = min($penaltyAmount, $depositLeft);
        $payableAmount = max( value: 0,  ...values: $penaltyAmount - $depositLeft);


        if ($depositDeducted < 0 || $payableAmount < 0) {
            throw ValidationException::withMessages([
                'penalty_amount' => 'Invalid penalty calculation.',
            ]);
        }

        // Combine as payload for pass to Service
        $payload = [...];

        // pass to service
        $payment = $this->service->createPenalty($payload);

        return response()->json([
            'timestamp' => $validated['timestamp'], // IFA timestamp
            'success' => true,
            'message' => 'Penalty payment created successfully.',
            'redirect_url' => route( name: 'admin.payments.list' PaymentController@list ),
            'data' => $payment,
```



```
        ]);

    } catch (\Throwable $e) {

        \Log::warning( message: '[storePenalty] failed', [
            'timestamp' => $validated['timestamp'], // IFA timestamp
            'error' => $e->getMessage(),
            'payload' => $payload,
        ]);

        return response()->json([
            'timestamp' => $validated['timestamp'], // IFA timestamp
            'success' => false,
            'message' => $e->getMessage(),
        ],  status: 400);
    }
}
```

Figure 5.2.4: Code Snippet of Error Handling for Store Penalty in Controller

- Class path: \app\Services\PaymentServices.php (function: createPenalty)



```
if (!$eventResponse->successful() || $eventResponse->json( key: 'status') !== 'success') {
    throw new RuntimeException( message: 'Failed to validate event.');
}

$eventMap = collect( value: $eventResponse->json( key: 'data') ?? []);
$eventData = $eventMap->get((string)$payload['event_id']);

if (!is_array($eventData)) {
    throw new InvalidArgumentException( message: 'Event not found.');
}

$eventOrganizerId = data_get($eventData,  key: 'organizer_id');

if (!$eventOrganizerId) {
    throw new RuntimeException( message: 'Event organizer not found.');
}
```

Figure 5.2.5: Code Snippet of Error Handling at Store Penalty Service

6. Web Services

1. Service Exposure

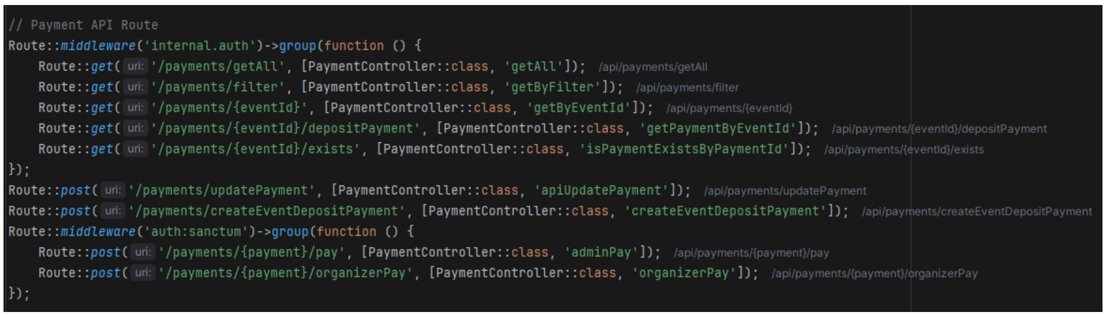
The payment module enables other system modules to access payment-related information in a standardized and loosely coupled manner by exposing RESTful web services. These services are implemented using JSON-based REST APIs, providing a lightweight and flexible communication approach suitable for modern web applications.

The exposed services are designed for modules such as the event module and user module to invoke for operations including payment validation, deposit verification, and penalty validation. All web services adhere to the Interface Protocol Agreement (IFA) standard, mandating request timestamps and maintaining a consistent JSON response structure to ensure traceability and interoperability.

Compared to SOAP, the RESTful architecture was selected for its simplicity, low overhead and cross-platform integration ease, making it more suitable for scalable, modular system design.

Team members use the API route in "\routes\api.php" for getting payment data needed.

Class Path: \routes\api\php



```php
// Payment API Route
Route::middleware('internal.auth')->group(function () {
    Route::get( uri: '/payments/getAll', [PaymentController::class, 'getAll']); /api/payments/getAll
    Route::get( uri: '/payments/filter', [PaymentController::class, 'getByFilter']); /api/payments/filter
    Route::get( uri: '/payments/{eventId}', [PaymentController::class, 'getByEventId']); /api/payments/{eventId}
    Route::get( uri: '/payments/{eventId}/depositPayment', [PaymentController::class, 'getPaymentByEventId']); /api/payments/{eventId}/depositPayment
    Route::get( uri: '/payments/{eventId}/exists', [PaymentController::class, 'isPaymentExistsByPaymentId']); /api/payments/{eventId}/exists
});
Route::post( uri: '/payments/updatePayment', [PaymentController::class, 'apiUpdatePayment']); /api/payments/updatePayment
Route::post( uri: '/payments/createEventDepositPayment', [PaymentController::class, 'createEventDepositPayment']); /api/payments/createEventDepositPayment
Route::middleware('auth:sanctum')->group(function () {
    Route::post( uri: '/payments/{payment}/pay', [PaymentController::class, 'adminPay']); /api/payments/{payment}/pay
    Route::post( uri: '/payments/{payment}/organizerPay', [PaymentController::class, 'organizerPay']); /api/payments/{payment}/organizerPay
});
```

Figure 6.1: API Route Provided from Payment Module

2. Webservice Mechanism

Get All Payments

	Description
Protocol	*RESTFUL*
Function Description	*Retrieve all payment records from the payment module, including associated event information and return the data in a structured JSON format compliant with the Interface Protocol Agreement (IFA) standard.*
Source Module	*Payment Module*

	Description
Target Module	*Payment Module*
URL	*http://127.0.0.1:8003/api/payments/getAll*
Function Name	*getAll*

Get Filtered Payments

	Description
Protocol	*RESTFUL*
Function Description	*Retrieves payment records that match the provided filter parameters such as payment ID, event ID, organizer ID, payment type, status and creation date range. The service returns filtered results in a standardized JSON response format.*
Source Module	*Payment Module*
Target Module	*Payment module, reward organizer after event, admin dashboard*
URL	*http://127.0.0.1:8003/api/payments/filter*
Function Name	*getByFilter*

Get Existing Payment by Event

	Description
Protocol	*RESTFUL*
Function Description	*Retrieves a list of payments filtered by event ID. The service returns all related payment records together with their associated event details, supporting event-level payment tracking and verification.*
Source Module	*Payment Module*
Target Module	*Event view detail*
URL	*http://127.0.0.1:8003/api/payments/{eventId}*
Function Name	*getByEventId*

Retrieve Event Deposit Payment

	Description
Protocol	*RESTFUL*
Function Description	*Retrieves the deposit payment record for a given event ID. If no deposit payment exists, the service returns a null data response. Only deposit-type payments are returned, excluding penalties or other payment types.*
Source Module	*Payment Module*
Target Module	*Event module*
URL	*http://127.0.0.1:8003/api/payments/{eventId}/depositPayment*
Function Name	*getPaymentByEventId*

Check Payment Existence by Event ID

	Description
Protocol	*RESTFUL*
Function Description	*Checks whether at least one payment record exists for the given event ID and returns a boolean result indicating the existence of payment data.*
Source Module	*Payment Module*
Target Module	*Event module*
URL	*http://127.0.0.1:8003/api/payments/{eventId}/exists*
Function Name	*isPaymentExistsByPaymentId*

Update Payment Status

	Description
Protocol	*RESTFUL*
Function Description	*Update the status of a specific payment based on the payment ID. Before executing the update, this service validates the requested status transition against configured rules, enforces payment type constraints and ensures refund conditions are met.*

	Description
Source Module	*Payment Module*
Target Module	*Payment Module, Event module*
URL	*http://127.0.0.1:8003/api/payments/updatePayment*
Function Name	*apiUpdatePayment*

Create Event Deposit Payment

	Description
Protocol	*RESTFUL*
Function Description	*Creates a new deposit payment for an event after validating that no existing active deposit payment exists. The service retrieves organizer membership information to calculate applicable discounts and computes the final payable amount before storing the payment record.*
Source Module	*Payment Module*
Target Module	*Event module*
URL	*http://127.0.0.1:8003/api/payments/createEventDepositPaymen t*
Function Name	*createEventDepositPayment*

Admin Mark Payment as Paid

	Description
Protocol	*RESTFUL*
Function Description	*Marks a pending payment as paid after validating the administrator role, payment status and selected payment method. For cash payments, the service verifies that the received amount is sufficient and automatically calculates any change before finalizing the payment.*
Source Module	*Payment Module*
Target Module	*Payment module*
URL	*http://127.0.0.1:8003/api/payments/{payment}/pay*

	Description
Function Name	*adminPay*

<u>Organizer Pay Payment by Card</u>

	Description
Protocol	*RESTFUL*
Function Description	*Processes a card payment for a pending payment initiated by an organizer. The service validates the organizer role, verifies payment ownership, and ensures that only card-based payments are accepted before marking the payment as paid.*
Source Module	*Payment Module*
Target Module	*Payment module*
URL	*http://127.0.0.1:8003/api/payments/{payment}/organizerPay*
Function Name	*organizerPay*

Web Services Request Parameter (provide)

<u>Get All Payments</u>

Field Name	**Field Type**	**Mandatory/ Optional**	**Description**	**Format**
timestamp	String	Mandatory	Timestamp used to identify and track the request in compliance with IFA requirements	ISO 8601 format such as YYYY-MM-DDTHH:MM:SS

<u>Get Payments by Filter</u>

Field Name	**Field Type**	**Mandatory/ Optional**	**Description**	**Format**
timestamp	String	Mandatory	Timestamp used to track	ISO 8601 format such as

			and audit the request in compliance with IFA requirements	YYYY-MM-DDTHH:MM:SS
id	Integer	Optional	Unique identifier of a specific payment	Numeric
event_id	Integer	Optional	Identifier of the event associated with the payment	Numeric
organizer_id	Integer	Optional	Identifier of the organizer who owns the payment	Numeric
type	String	Optional	Type of payment to be filtered	deposit, penalty, rental
status	String	Optional	Payment status to be filtered	pending, paid, overdue, cancelled, refunded
created_at_start	String	Optional	Start date for filtering payments by creation time	YYYY-MM-DD
created_at_end	String	Optional	End date for filtering payments by creation time	YYYY-MM-DD

Get Payments by Event ID

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Timestamp used to track	ISO 8601 format such as

			and audit the request in compliance with IFA requirements	YYYY-MM-DDTHH:MM:SS
eventId	Integer	Mandatory	Unique identifier of the event used to retrieve all related payments	Numeric

Check Payment Exists by Event ID

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Timestamp used to track and audit the request in compliance with IFA requirements	ISO 8601 format such as YYYY-MM-DDTHH:MM:SS
eventId	Integer	Mandatory	Unique identifier of the event used to retrieve all related payments	Numeric

Check Payment Exists by Event ID

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Timestamp used to track and audit the request in compliance with IFA	ISO 8601 format such as YYYY-MM-DDTHH:MM:SS

			requirements	
eventId	Integer	Mandatory	Unique identifier of the event used to retrieve all related payments	Numeric

Update Payment Status

Field Name	**Field Type**	**Mandatory/ Optional**	**Description**	**Format**
timestamp	String	Mandatory	Timestamp used to track and audit the request in compliance with IFA requirements	ISO 8601 format such as YYYY-MM-DD THH:MM:SS
payment_id	Integer	Mandatory	Unique identifier of the payment to be updated	Numeric
status	String	Mandatory	New payment status to be applied	pending, cancelled, overdue, refunded

Create Event Deposit Payment

Field Name	**Field Type**	**Mandatory/ Optional**	**Description**	**Format**
timestamp	String	Mandatory	Timestamp used to track and audit the request in compliance with IFA requirements	ISO 8601 format such as YYYY-MM-DD THH:MM:SS

event_id	Integer	Mandatory	Unique identifier of the event that requires a deposit payment	Numeric
deposit_amt	Decimal	Mandatory	Original deposit amount before membership discount is applied	Numeric (≥ 0)
organizer_id	Integer	Mandatory	Unique identifier of the organizer responsible for the deposit	Numeric

Admin Pay Payment

Field Name	**Field Type**	**Mandatory/ Optional**	**Description**	**Format**
timestamp	String	Mandatory	Timestamp used for request tracking and IFA compliance	ISO 8601 format such as YYYY-MM-DD THH:MM:SS
payment	Integer (Path Param)	Mandatory	Unique identifier of the payment to be marked as paid	Numeric
payment_method	String	Mandatory	Payment method used by the admin	cash, pos
received_amt	Decimal	Optional	Actual amount received (required when payment	Numeric (≥ 0)

			method is cash)	

<u>Organizer Pay Payment</u>

Field Name	Field Type	Mandatory/ Optional	Description	Format
timestamp	String	Mandatory	Timestamp used for request tracking and IFA compliance	ISO 8601 format such as YYYY-MM-DD THH:MM:SS
payment	Integer (Path Param)	Mandatory	Unique identifier of the payment to be paid by the organizer	Numeric
payment_method	String	Mandatory	Payment method selected by the organizer	card
card_number	String	Mandatory	Card number used for payment	13–19 digits
ccv	String	Mandatory	Card security code	3–4 digits
expired_date	String	Mandatory	Card expiration date	YYYY-MM-DD

Web Services Response Parameter (consume)

<u>Bulk Event Retrieval</u>

Field Name	Field Type	Mandatory/ Optional	Description	Format
status	String	Mandatory	Status of the request	success or error

timestamp	String	Mandatory	Time when the response was generated	ISO 8601
data	Object	Mandatory	Contains the details that are needed from the query	JSON Object Event ID: Unique identifier of the event Event Name: Name of the event Event Description: Description of the event Event Category: Category information of the event Event Schedule: Event start and end date Event Status: Current status of the event Deposit Requirement: Indicates whether deposit payment is required Organizer Information: Organizer details associated with the event Proposal Information: Uploaded proposal metadata Event Images: Event-related

				images

Penalty Eligible Event Retrieval

Field Name	Field Type	Mandatory/ Optional	Description	Format
status	String	Mandatory	Status of the request	success or error
timestamp	String	Mandatory	Time when the response was generated	ISO 8601
data	Array	Mandatory	Contains the details of events eligible for penalty processing	JSON Array Event ID: Unique identifier of the event Event Name: Name of the event Event Description: Description of the event Event Category: Category information of the event Event Schedule: Event start and end date Event Status: Current status of the event Deposit Requirement: Indicates whether deposit payment is required

				Organizer Information: Organizer details associated with the event Proposal Information: Uploaded proposal metadata Event Images: Event-related images

Update Event Status

Field Name	**Field Type**	**Mandatory/ Optional**	**Description**	**Format**
status	String	Mandatory	Status of the request	success or failed
timestamp	String	Mandatory	Time when the response was generated	ISO 8601
message	String	Mandatory	Result message of the event status update operation	Text

Organizer Blacklist Status Update

Field Name	**Field Type**	**Mandatory/ Optional**	**Description**	**Format**
status	String	Mandatory	Status of the request	success or error
timestamp	String	Mandatory	Time when the response was	ISO 8601

			generated	
message	String	Mandatory	Result message of the blacklist status update operation	Text
updated	Boolean	Mandatory	Indicates whether the organizer blacklist status is successfully updated	true or false

Organizer Credit Score Update

Field Name	**Field Type**	**Mandatory/ Optional**	**Description**	**Format**
status	String	Mandatory	Status of the request	success or error
timestamp	String	Mandatory	Time when the response was generated	ISO 8601
message	String	Mandatory	Result message of the blacklist status update operation	Text

Get Organizer Membership

Field Name	**Field Type**	**Mandatory/ Optional**	**Description**	**Format**
status	String	Mandatory	Status of the request	success or error
timestamp	String	Mandatory	Time when the response was generated	ISO 8601

message	String	Mandatory	Error message when request fails	Text
data	Object	Conditional	Contains membership details when request is successful	Object Membership ID: Unique identifier of the membership assigned to the organizer. Membership Name: Name of the membership level. Discount Rate: Discount percentage applied to payments made by the organizer. Credit Threshold: Minimum credit score required to maintain the membership status. Membership Status: Indicates whether the membership is currently active or inactive.

3. Service Consumption

The payment module invokes web services provided by other system modules to support payment verification, penalty processing and membership discount calculation functions. This module retrieves required information through standardized RESTful APIs, preventing redundant data storage across modules. This approach ensures data consistency and reduces tight coupling between system components. All invoked services comply with Interface Framework Agreement (IFA) requirements, incorporating mandatory timestamps and structured response formats.

<u>**Event Module**</u>

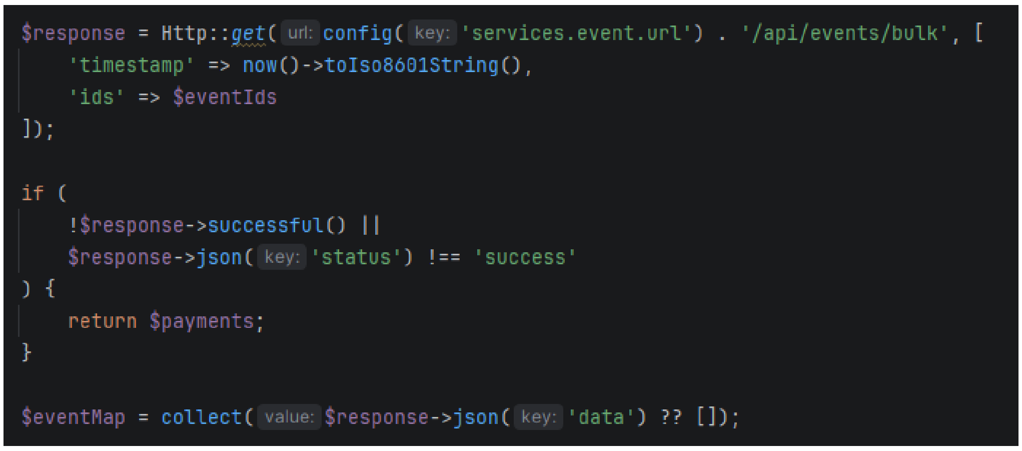
The payment module calls the event module's web service to retrieve event-related information, including event status, organizer affiliation, event schedule and deposit requirements. This information is used to validate deposit payments, determine eligibility for penalty fees and enforce refund and late payment rules. By designating the event module as the authoritative source for event data, the payment module ensures financial operations are consistently based on accurate and up-to-date event information while maintaining clear separation of duties.

Example:

1. GET (Retrieve Event Details)

   The payment module sends a GET request to the event module's "/events/bulk" API to retrieve event information associated with payment records. This enables payment data to be enriched with the latest event details while avoiding redundant storage of event data across modules.

   Class path: \app\Http\Controller\PaymentController.php (function: attachEventsToPayments)



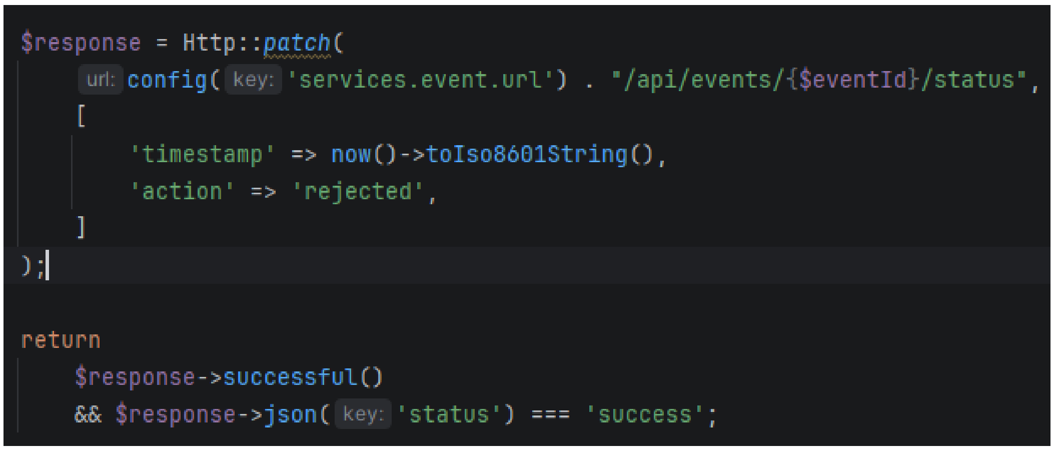
```
$response = Http::get( url: config( key: 'services.event.url') . '/api/events/bulk', [
    'timestamp' => now()->toIso8601String(),
    'ids' => $eventIds
]);

if (
    !$response->successful() ||
    $response->json( key: 'status') !== 'success'
) {
    return $payments;
}

$eventMap = collect( value: $response->json( key: 'data') ?? []);
```

Figure 6.3.1: Code Snippet of Using GET to Call Event's API

2. PATCH (Update Event Status)

   When a deposit payment is overdue, the payment module sends a PATCH request to the event module's "/events/{eventId}/status API" to reject the associated event. This ensures that unpaid or overdue payments are consistently reflected throughout the event lifecycle management process.

   Class path: \app\Http\Controller\PaymentController.php (function: rejectEvent)



```
$response = Http::patch(
    url: config( key: 'services.event.url') . "/api/events/{$eventId}/status",
    [
        'timestamp' => now()->toIso8601String(),
        'action' => 'rejected',
    ]
);

return
    $response->successful()
    && $response->json( key: 'status') === 'success';
```

Figure 6.3.2: Code Snippet of Using PATCH to Call Event's API

User and Membership Module

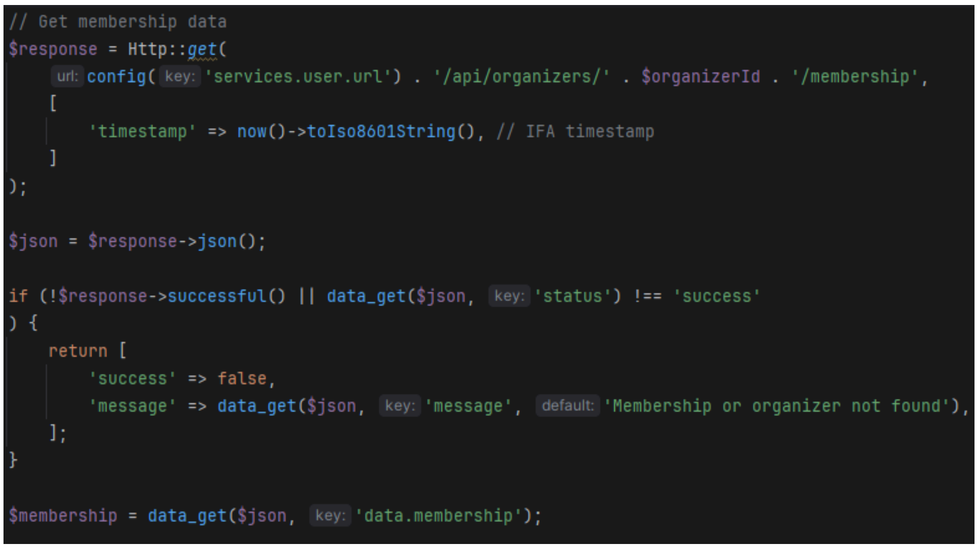
Additionally, the payment module invokes services from the user and membership modules to support membership-related financial logic. When creating payments, membership information is used to calculate deposit discounts. When penalties or late payments occur, services for updating credit scores and blacklist statuses are invoked. This approach centralizes user and membership management within the user and membership modules, avoiding duplication of business logic while ensuring consistent enforcement of membership rules.

Example:

1. GET (Retrieve Organizer Membership Information)

   The payment module sends a GET request to the user and membership modules when creating a payment to retrieve the organizer's membership details. This information is used to precisely apply membership-based discounts without requiring local storage of membership data.

   Class path: \app\Http\Controller\PaymentController.php (function: createEventDepositPayment)



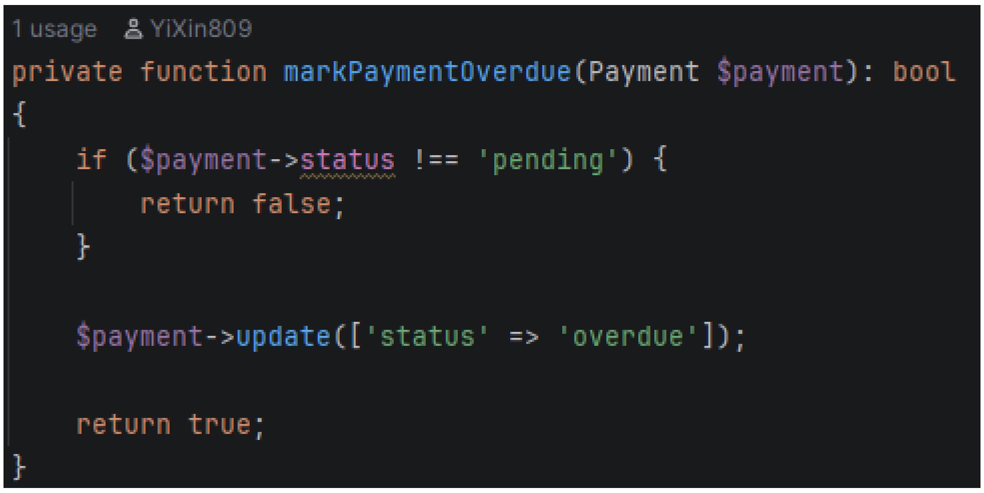
```
// Get membership data
$response = Http::get(
    url: config( key: 'services.user.url') . '/api/organizers/' . $organizerId . '/membership',
    [
        'timestamp' => now()->toIso8601String(), // IFA timestamp
    ]
);

$json = $response->json();

if (!$response->successful() || data_get($json,  key: 'status') !== 'success'
) {
    return [
        'success' => false,
        'message' => data_get($json,  key: 'message',  default: 'Membership or organizer not found'),
    ];
}

$membership = data_get($json,  key: 'data.membership');
```

Figure 6.3.4: Code Snippet of Using GET to Call User's API

2. POST (Update Organizer Blacklist Status)

   When a deposit payment is overdue, the payment module sends a PATCH request to the event module's "/events/{eventId}/status API" to reject the associated event. This ensures that unpaid or overdue payments are consistently reflected throughout the event lifecycle management process.

   Class path: \app\Http\Controller\PaymentController.php (function: markPaymentOverdue)



```
1 usage   YiXin809
private function markPaymentOverdue(Payment $payment): bool
{
    if ($payment->status !== 'pending') {
        return false;
    }

    $payment->update(['status' => 'overdue']);

    return true;
}
```

Figure 6.3.4: Code Snippet of Using POST to Call User's API

7. Index

2.4.1	-	Class Path: \app\Http\Controller/PaymentController.php (function: createEventDepositPayment)
2.4.2	-	\app\Services\PaymentService.php (function: createEventDepositPayment)
2.5.1	\resources\views\livewire\admin\penalty\create-penalty-payment.blade.php	\app\Livewire\Admin\Penalty\CreatePenaltyPayment.php
2.6.1	\resources\views\livewire\admin\make-payment.blade.php	\app\Livewire\Admin\MakePayment.php
2.7.1	\resources\views\livewire\organizer\make-payment.blade.php	\app\Livewire\Organizer\MakePayment.php
2.8.1	/resources/views/livewire/organizer/organizer-payment-table.blade.php	\app\Livewire\Organizer\OrganizerPaymentTable.php
2.8.2	-	\app\Http\Controller\PaymentControlelr.php (function: receipt)
2.9.1	-	\app\Console\Command\CheckOverduePayment.php
2.9.2	-	\app\Services\PaymentServices.php (function: handleOverduePayments)
3.1	-	-
4.2	-	-
5.2.1	-	/app/Livewire/Admin/PaymentTable.php (function: list)
5.2.2	-	\app\Http\Middleware\InternalApiAuth.php
5.2.3	-	\routes\api.php
5.2.4	-	\app\Http\Controller\PaymentContrololr.php (function: storePenalty)
5.2.5	-	\app\Services\PaymentServices.php (function: createPenalty)

6.1	-	\routes\api\php
6.3.1	-	\app\Http\Controller\PaymentController.php (function: attachEventsToPayments)
6.3.2	-	\app\Http\Controller\PaymentController.php (function: rejectEvent)
6.3.3	-	\app\Http\Controller\PaymentController.php (function: createEventDepositPayment)
6.3.4	-	\app\Http\Controller\PaymentController.php (function: markPaymentOverdue)

8. References

OWASP. “OWASP Top 10 API Security Risks – 2023.” Owasp.org, 2023, owasp.org/API-Security/editions/2023/en/0x11-t10/. Accessed 17 Dec. 2025.

“Error Handling - OWASP Cheat Sheet Series.” Cheatsheetseries.owasp.org, cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html. Accessed 17 Dec. 2025.

Refactoring Guru. “Observer.” Refactoring.guru, 2014, refactoring.guru/design-patterns/observer. Accessed 18 Dec. 2025.\

“Eloquent: Getting Started - Laravel 12.x - the PHP Framework for Web Artisans.” Laravel.com, 2025, laravel.com/docs/12.x/eloquent#observers. Accessed 18 Dec. 2025.

Martin Fowler. “What Do You Mean by “Event-Driven”?” Martinfowler.com, 7 Feb. 2017, martinfowler.com/articles/201701-event-driven.html. Accessed 18 Dec. 2025.

Lewis, James, and Martin Fowler. “Microservices.” Martinfowler.com, 25 Mar. 2014, martinfowler.com/articles/microservices.html. Accessed 16 Dec. 2025.