Practical Examples

Product Module Examples

1. VPS Hosting Module

Complete implementation of a VPS hosting service module with cloud provider integration.

Main Module Class

```
<?php

use App\Models\Service;
use App\Models\Task;
use App\Modules\Product;
use Illuminate\Support\Facades\Validator;
use Modules\Product\VPSHosting\Services\CloudAPIClient;

class VPSHosting extends Product
{
    private CloudAPIClient $apiClient;

    public function __construct()
    {
        parent::__construct();
        $this->apiClient = new CloudAPIClient(
            $this->config('api_url'),
            $this->config('api_key')
        );
    }

    public function activate(): string
    {
        try {
            Schema::create('vps_servers', function (Blueprint $table) {
                $table->id();
```

```
                $table->uuid('service_uuid');
                $table->string('server_id')->nullable();
                $table->ipAddress('ip_address')->nullable();
                $table->string('hostname');
                $table->string('root_password');
                $table->enum('status', ['creating', 'active', 'suspended',
'terminated']);
                $table->json('server_specs');
                $table->timestamps();

                $table->foreign('service_uuid')->references('uuid')-
>on('services');
                $table->index(['service_uuid', 'status']);
            });

            Schema::create('vps_images', function (Blueprint $table) {
                $table->id();
                $table->string('image_id');
                $table->string('name');
                $table->string('os_family');
                $table->string('version');
                $table->boolean('active')->default(true);
                $table->timestamps();
            });

            // Seed default images
            $this->seedDefaultImages();

            return 'success';
        } catch (Exception $e) {
            $this->logError('activate', $e->getMessage());
            return 'Error: ' . $e->getMessage();
        }
    }

    public function getProductData(array $data = []): array
    {
        $this->product_data = [
            'cpu_cores' => $data['cpu_cores'] ?? 1,
            'ram_mb' => $data['ram_mb'] ?? 1024,
```

```
            'disk_gb' => $data['disk_gb'] ?? 25,
            'bandwidth_gb' => $data['bandwidth_gb'] ?? 1000,
            'location' => $data['location'] ?? 'us-east-1',
            'backup_enabled' => $data['backup_enabled'] ?? false,
        ];
        return $this->product_data;
    }

    public function saveProductData(array $data = []): array
    {
        $validator = Validator::make($data, [
            'cpu_cores' => 'required|integer|min:1|max:32',
            'ram_mb' => 'required|integer|min:512|max:65536',
            'disk_gb' => 'required|integer|min:10|max:1000',
            'bandwidth_gb' => 'required|integer|min:100',
            'location' => 'required|in:us-east-1,us-west-1,eu-west-1',
            'backup_enabled' => 'boolean',
        ]);

        if ($validator->fails()) {
            return [
                'status' => 'error',
                'message' => $validator->errors(),
                'code' => 422,
            ];
        }

        return ['status' => 'success', 'data' => $data, 'code' => 200];
    }

    public function getServiceData(array $data = []): array
    {
        $this->service_data = [
            'hostname' => $data['hostname'] ?? '',
            'image_id' => $data['image_id'] ?? 'ubuntu-20.04',
            'ssh_keys' => $data['ssh_keys'] ?? [],
            'root_password' => $data['root_password'] ?? $this-
>generateSecurePassword(),
        ];
        return $this->service_data;
```

```
    }

    public function createJob(): array
    {
        try {
            $service = Service::find($this->service_uuid);
            $service->setProvisionStatus('processing');

            // Create VPS through cloud API
            $vpsData = [
                'name' => $this->service_data['hostname'],
                'image' => $this->service_data['image_id'],
                'size' => $this->getServerSize(),
                'region' => $this->product_data['location'],
                'ssh_keys' => $this->service_data['ssh_keys'],
            ];

            $result = $this->apiClient->createServer($vpsData);

            if ($result['success']) {
                // Store server details
                DB::table('vps_servers')->insert([
                    'service_uuid' => $this->service_uuid,
                    'server_id' => $result['data']['id'],
                    'hostname' => $this->service_data['hostname'],
                    'root_password' => encrypt($this->service_data['root_password']),
                    'status' => 'creating',
                    'server_specs' => json_encode($this->product_data),
                    'created_at' => now(),
                    'updated_at' => now(),
                ]);

                                        // Queue status check job
                $this->scheduleStatusCheck($result['data']['id']);

                $this->logInfo('createJob', 'VPS creation initiated', $result);
                return ['status' => 'success'];
            } else {
```

```
                $service->setProvisionStatus('failed');
                $this->logError('createJob', 'VPS creation failed', $result);
                return ['status' => 'error', 'message' => $result['error']];
            }
        } catch (Exception $e) {
            $service->setProvisionStatus('failed');
            $this->logError('createJob', 'Exception in VPS creation', $e->getMessage());
            return ['status' => 'error', 'message' => 'VPS creation failed'];
        }
    }

    private function getServerSize(): string
    {
        // Map product specs to cloud provider size
        $cpu = $this->product_data['cpu_cores'];
        $ram = $this->product_data['ram_mb'];

        if ($cpu == 1 && $ram <= 1024) return 's-1vcpu-1gb';
        if ($cpu == 2 && $ram <= 2048) return 's-2vcpu-2gb';
        if ($cpu == 4 && $ram <= 8192) return 's-4vcpu-8gb';

        return 'custom';
    }

    private function scheduleStatusCheck(string $serverId): void
    {
        // Queue a job to check server status in 30 seconds
        $data = [
            'module' => $this,
            'method' => 'statusCheckJob',
            'server_id' => $serverId,  // Store as additional data
            'tries' => 2,
            'backoff' => 30,
            'timeout' => 120,
            'maxExceptions' => 1,
        ];

        Task::add('ModuleJob', 'Module', $data, ['status_check', 'vps']);
    }
```

```
    public function statusCheckJob(): array
    {
        try {
            // Get server ID from VPS servers table
            $vpsServer = DB::table('vps_servers')
                ->where('service_uuid', $this->service_uuid)
                ->first();

            if (!$vpsServer) {
                $this->logError('statusCheckJob', 'VPS server record not
found');
                return ['status' => 'error'];
            }

            $result = $this->apiClient->getServerStatus($vpsServer-
>server_id);

            if ($result['success'] && $result['status'] === 'active') {
                $service = Service::find($this->service_uuid);
                $service->setProvisionStatus('completed');

                // Update server IP if available
                if (!empty($result['ip_address'])) {
                    DB::table('vps_servers')
                        ->where('service_uuid', $this->service_uuid)
                        ->update([
                            'ip_address' => $result['ip_address'],
                            'status' => 'active',
                            'updated_at' => now(),
                        ]);
                }

                $this->logInfo('statusCheckJob', 'VPS is now active', $result);
            } else {
                $this->logInfo('statusCheckJob', 'VPS still provisioning',
$result);

                // Re-queue status check if still creating
                if ($result['status'] === 'new' || $result['status'] === 'creating')
```

```
{
                    $this->scheduleStatusCheck($vpsServer->server_id);
                }
            }

            return ['status' => 'success'];
        } catch (Exception $e) {
            $this->logError('statusCheckJob', 'Status check failed', $e->getMessage());
            return ['status' => 'error'];
        }
    }

    public function getClientAreaMenuConfig(): array
    {
        return [
            'general' => [
                'name' => 'Overview',
                'template' => 'client_area.overview',
            ],
            'console' => [
                'name' => 'Console',
                'template' => 'client_area.console',
            ],
            'snapshots' => [
                'name' => 'Snapshots',
                'template' => 'client_area.snapshots',
            ],
            'monitoring' => [
                'name' => 'Monitoring',
                'template' => 'client_area.monitoring',
            ],
        ];
    }

    public function controllerClient_consoleGet(Request $request): JsonResponse
    {
        try {
            $vpsServer = DB::table('vps_servers')
                ->where('service_uuid', $this->service_uuid)
```

```
                ->first();

            if (!$vpsServer) {
                return response()->json(['error' => 'Server not found'], 404);
            }

            $consoleUrl = $this->apiClient->getConsoleUrl($vpsServer->server_id);

            return response()->json([
                'success' => true,
                'console_url' => $consoleUrl,
            ]);
        } catch (Exception $e) {
            return response()->json(['error' => 'Console unavailable'], 500);
        }
    }
}
```

Cloud API Client Service

```
<?php

namespace Modules\Product\VPSHosting\Services;

use GuzzleHttp\Client;
use GuzzleHttp\Exception\GuzzleException;

class CloudAPIClient
{
    private Client $client;
    private string $apiKey;

    public function __construct(string $baseUrl, string $apiKey)
    {
        $this->apiKey = $apiKey;
        $this->client = new Client([
            'base_uri' => $baseUrl,
            'timeout' => 30,
            'headers' => [
                'Authorization' => 'Bearer ' . $apiKey,
```

```
                'Content-Type' => 'application/json',
            ],
        ]);
    }

    public function createServer(array $data): array
    {
        try {
            $response = $this->client->post('/v2/droplets', [
                'json' => $data,
            ]);

            $result = json_decode($response->getBody(), true);

            return [
                'success' => true,
                'data' => $result['droplet'],
            ];
        } catch (GuzzleException $e) {
            return [
                'success' => false,
                'error' => 'API Error: ' . $e->getMessage(),
            ];
        }
    }

    public function getServerStatus(string $serverId): array
    {
        try {
            $response = $this->client->get("/v2/droplets/{$serverId}");
            $result = json_decode($response->getBody(), true);

            return [
                'success' => true,
                'status' => $result['droplet']['status'],
                'ip_address' => $result['droplet']['networks']['v4'][0]['ip_address'] ??
null,
            ];
        } catch (GuzzleException $e) {
```

```
            return ['success' => false, 'error' => $e->getMessage()];
        }
    }

    public function powerAction(string $serverId, string $action): array
    {
        try {
            $response = $this->client->post("/v2/droplets/{$serverId}/actions",
[
                'json' => ['action' => $action],
            ]);

            return ['success' => true];
        } catch (GuzzleException $e) {
            return ['success' => false, 'error' => $e->getMessage()];
        }
    }
}
```

Plugin Module Examples

1. Monitoring Plugin

```
<?php

use App\Modules\Plugin;

class ServerMonitoring extends Plugin
{
    public function activate(): string
    {
        try {
            Schema::create('monitoring_checks', function (Blueprint $table) {
                $table->id();
                $table->string('name');
                $table->string('type'); // ping, http, port
                $table->string('target'); // IP/URL to monitor
                $table->json('config'); // Check-specific config
```

```
                $table->integer('interval')->default(60); // seconds
                $table->boolean('active')->default(true);
                $table->timestamps();
            });

            Schema::create('monitoring_results', function (Blueprint $table) {
                $table->id();
                $table->foreignId('check_id')-
>constrained('monitoring_checks');
                $table->boolean('status'); // up/down
                $table->integer('response_time')->nullable(); // milliseconds
                $table->string('error_message')->nullable();
                $table->timestamp('checked_at');
                $table->timestamps();

                $table->index(['check_id', 'checked_at']);
            });

                    // Schedule monitoring job
        $this->scheduleMonitoringJob();

        $this->logInfo('activate', 'Monitoring plugin activated successfully');

            return 'success';
        } catch (Exception $e) {
            return 'Error: ' . $e->getMessage();
        }
    }

    public function adminWebRoutes(): array
    {
        return [
            [
                'method' => 'get',
                'uri' => 'dashboard',
                'permission' => 'monitoring-dashboard',
                'name' => 'dashboard',
                'controller' => 'ServerMonitoringController@dashboard',
            ],
```

```
            [
                'method' => 'get',
                'uri' => 'checks',
                'permission' => 'monitoring-checks',
                'name' => 'checks',
                'controller' => 'ServerMonitoringController@checks',
            ],
        ];
    }

    private function scheduleMonitoringJob(): void
    {
        // Add recurring monitoring task
        $data = [
            'module' => $this,
            'method' => 'runChecks',
            'tries' => 2,
            'backoff' => 60,
            'timeout' => 300,
            'maxExceptions' => 1,
        ];

        Task::add('ModuleJob', 'Module', $data, ['monitoring', 'recurring']);
    }

    public function runChecks(): array
    {
        $checks = DB::table('monitoring_checks')
            ->where('active', true)
            ->get();

        foreach ($checks as $check) {
            $this->executeCheck($check);
        }

        return ['status' => 'success'];
    }

    private function executeCheck($check): void
```

```
    {
        $startTime = microtime(true);
        $success = false;
        $errorMessage = null;

        try {
            switch ($check->type) {
                case 'ping':
                    $success = $this->pingCheck($check->target);
                    break;
                case 'http':
                    $success = $this->httpCheck($check->target, json_decode($check->config,
true));
                    break;
                case 'port':
                    $success = $this->portCheck($check->target, json_decode($check->config,
true));
                    break;
            }
        } catch (Exception $e) {
            $errorMessage = $e->getMessage();
        }

        $responseTime = round((microtime(true) - $startTime) * 1000);

        DB::table('monitoring_results')->insert([
            'check_id' => $check->id,
            'status' => $success,
            'response_time' => $responseTime,
            'error_message' => $errorMessage,
            'checked_at' => now(),
            'created_at' => now(),
            'updated_at' => now(),
        ]);

        // Send alerts if needed
        if (!$success) {
            $this->sendAlert($check, $errorMessage);
        }
```

```
    }
}
```

Payment Module Examples

Bank Transfer Module

```
<?php

use App\Modules\Payment;

class BankTransfer extends Payment
{
    public function getClientAreaHtml(array $data = []): string
    {
        $invoice = $data['invoice'];
        $amount = $invoice->getDueAmountAttribute();
        $currency = $invoice->client->currency->code;

        // Generate unique reference number
        $reference = 'BT' . strtoupper(substr(md5($invoice->uuid . time()), 0, 8));

        // Store payment instruction
        DB::table('bank_transfer_payments')->insert([
            'invoice_uuid' => $invoice->uuid,
            'amount' => $amount,
            'currency' => $currency,
            'reference' => $reference,
            'status' => 'awaiting_transfer',
            'expires_at' => now()->addDays(7),
            'created_at' => now(),
        ]);

        return $this->view('client_area', [
            'reference' => $reference,
            'bank_details' => $this->getBankDetails($currency),
            'amount' => $amount,
            'currency' => $currency,
```

```
            'instructions' => $this->getTransferInstructions($reference),
        ]);
    }

    private function getBankDetails(string $currency): array
    {
        $bankDetails = [
            'USD' => [
                'bank_name' => 'Example Bank USA',
                'account_name' => 'YourCompany LLC',
                'account_number' => '1234567890',
                'routing_number' => '021000021',
                'swift_code' => 'EXBKUS33',
            ],
            'EUR' => [
                'bank_name' => 'Example Bank Europe',
                'account_name' => 'YourCompany EU',
                'iban' => 'DE89370400440532013000',
                'bic' => 'EXBKDEFF',
            ],
        ];

        return $bankDetails[$currency] ?? $bankDetails['USD'];
    }
}
```

Notification Module Examples

Slack Notification Module

```
<?php

use App\Modules\Notification;
use GuzzleHttp\Client;

class SlackNotification extends Notification
{
    private Client $client;
```

```
    public function __construct()
    {
        parent:: __construct();
        $this->client = new Client(['timeout' => 10]);
    }

    public function send(array $data): array
    {
        try {
            $webhookUrl = $this->module_data['webhook_url'] ?? '';

            if (empty($webhookUrl)) {
                return ['status' => 'error', 'message' => 'Webhook URL not
configured'];
            }

            $message = $this->buildSlackMessage($data);

            $response = $this->client->post($webhookUrl, [
                'json' => $message,
            ]);

            if ($response->getStatusCode() === 200) {
                $this->logInfo('send', 'Slack notification sent', $data);
                return ['status' => 'success'];
            } else {
                $this->logError('send', 'Slack API error', [
                    'status_code' => $response->getStatusCode(),
                    'body' => $response->getBody()->getContents()
                ]);
                return ['status' => 'error', 'message' => 'Slack API error'];
            }

        } catch (Exception $e) {
            $this->logError('send', 'Slack notification failed', $e-
>getMessage());
            return ['status' => 'error', 'message' => $e->getMessage()];
        }
    }
```

```
    private function buildSlackMessage(array $data): array
    {
        $color = $this->getColorForType($data['type']);

        return [
            'attachments' => [
                [
                    'color' => $color,
                    'title' => $data['title'],
                    'text' => $data['message'],
                    'fields' => [
                        [
                            'title' => 'Time',
                            'value' => now()->toDateTimeString(),
                            'short' => true,
                        ],
                        [
                            'title' => 'Module',
                            'value' => $data['module'] ??
'System',
                            'short' => true,
                        ],
                    ],
                    'footer' => 'PUQcloud',
                    'ts' => time(),
                ],
            ],
        ];
    }

    private function getColorForType(string $type): string
    {
        return match($type) {
            'error' => 'danger',
            'warning' => 'warning',
            'success' => 'good',
            default => '#36a64f',
        };
    }
```

```
}
```

These examples demonstrate real-world implementations of different module types, showing how to integrate with external APIs, handle complex business logic, and implement robust error handling. Each example follows the established patterns while showcasing specific use cases and best practices.

Revision #3
Created 30 July 2025 02:35:00 by Dmytro Kravchenko
Updated 13 August 2025 05:11:33 by Dmytro Kravchenko