Development of modules

This section will describe in detail how modules for PUQcloud Panel are created — from the structure to integration with the panel, logic, interface and events. The section is designed for developers who want to expand the functionality of the panel to suit their own needs.

- Module Development Guide
- API Reference
- Practical Examples

Module Development Guide

Architecture Overview

Module Framework Architecture

PUQcloud uses a hierarchical module system based on inheritance and dynamic loading:

```
Base Module Class
├── Product Module
├── Plugin Module
├── Payment Module
└── Notification Module
```

Core Components

- **Module Class:** Main business logic and lifecycle management
- **Controllers:** Handle HTTP requests and API endpoints
- **Models:** Database interactions and data validation
- **Views:** Blade templates for UI rendering
- **Configuration:** Module metadata and settings
- **Task System:** Asynchronous job processing

Module Types

Type	Purpose	Examples
Product	Service provisioning and management	VPS, Cloud Services
Plugin	System extensions and integrations	Monitoring, Analytics
Payment	Payment gateway integrations	Stripe, PayPal, Bank transfers
Notification	Communication channels	Email, SMS, Slack

Getting Started

Prerequisites

- **PHP 8.1+**
- **Laravel 9.0+**
- **Composer**
- **Redis/Database for queues**
- **Basic understanding of Laravel concepts**

Creating Your First Module

Step 1: Choose Module Type and Name

```
# Module naming convention: StudlyCaseModuleName (exact filename match required)
# Directory structure: modules/{Type}/{ModuleName}/
mkdir -p modules/Product/MyCloudService
```

Step 2: Create Professional Module Structure

Based on real PUQcloud modules like puqNextcloud:

```
modules/Product/MyCloudService/
├── MyCloudService.php            # Main module class (required)
├── config.php                    # Module metadata (required)
├── hooks.php                     # Event hooks (optional)
├── Controllers/                  # HTTP request handlers
│   └── MyCloudServiceController.php
├── Services/                     # External API clients
│   └── CloudAPIClient.php
├── Models/                       # Database models
│   ├── CloudServer.php
│   └── CloudServerGroup.php
├── views/                        # User interface templates
│   ├── admin_area/              # Admin panel views
│   │   ├── product.blade.php
│   │   ├── service.blade.php
│   │   ├── servers.blade.php
│   │   └── configuration.blade.php
│   ├── client_area/             # Client panel views
│   │   ├── general.blade.php
│   │   ├── files.blade.php
```

```
│   │   └── statistics.blade.php
│   └── assets/                  # Static resources
│       ├── css/
│       ├── js/
│       └── img/
└── lang/                        # Internationalization
    ├── en.php
    └── pl.php
```

Module Structure

Main Module Class

```php
<?php

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

class MyCloudService extends Product
{
    public $product_data;
    public $product_uuid;
    public $service_data;
    public $service_uuid;

    public function __construct()
    {
        parent::__construct();
    }

    // Lifecycle Methods
    public function activate(): string
    {
        try {
            // Create necessary database tables
            $this->createTables();
            $this->logInfo('activate', 'Module activated successfully');
```

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

    public function deactivate(): string
    {
        try {
            // Cleanup resources
            $this->dropTables();
            $this->logInfo('deactivate', 'Module deactivated successfully');
            return 'success';
        } catch (Exception $e) {
            $this->logError('deactivate', 'Deactivation failed: ' . $e->getMessage());
            return 'error: ' . $e->getMessage();
        }
    }

    // Product Configuration
    public function getProductData(array $data = []): array
    {
        $this->product_data = [
            'server_location' => $data['server_location'] ?? '',
            'plan_type' => $data['plan_type'] ?? '',
            'disk_space' => $data['disk_space'] ?? '',
            'bandwidth' => $data['bandwidth'] ?? '',
        ];
        return $this->product_data;
    }

    public function saveProductData(array $data = []): array
    {
        $validator = Validator::make($data, [
            'server_location' => 'required|string',
            'plan_type' => 'required|in:basic,premium,enterprise',
            'disk_space' => 'required|integer|min:1',
            'bandwidth' => 'required|integer|min:1',
```

```
        ]);

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

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

    // Service Management
    public function getServiceData(array $data = []): array
    {
        $this->service_data = [
            'domain' => $data['domain'] ?? '',
            'username' => $data['username'] ?? '',
            'password' => $data['password'] ?? '',
            'ip_address' => $data['ip_address'] ?? '',
        ];
        return $this->service_data;
    }

    public function saveServiceData(array $data = []): array
    {
        $validator = Validator::make($data, [
            'domain' => 'required|string|max:255',
            'username' => 'required|string|max:50',
            'password' => 'required|string|min:8',
            'ip_address' => 'nullable|ip',
        ]);

        if ($validator->fails()) {
            return [
```

```
            'status' => 'error',
            'message' => $validator->errors(),
            'code' => 422,
        ];
    }

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

// Asynchronous Operations
public function create(): array
{
    $data = [
        'module' => $this,
        'method' => 'createJob',
        'tries' => 1,
        'backoff' => 60,
        'timeout' => 600,
        'maxExceptions' => 1,
    ];

    $service = Service::find($this->service_uuid);
    $service->setProvisionStatus('processing');
    Task::add('ModuleJob', 'Module', $data, ['create']);

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

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

        // Generate service credentials (real implementation pattern)
        $this->service_data['username'] = $this->product_data['username_prefix']
```

```
.
                                            random_int(100000, 999999)
.
                                            $this-
>product_data['username_suffix'];
            $this->service_data['password'] =
generateStrongPassword(10);

            // Create API client and provision account
            $apiClient = new HostingAPIClient($this->config('api_url'), $this-
>config('api_key'));
            $result = $apiClient->createAccount([
                'domain' => $this->service_data['domain'],
                'username' => $this->service_data['username'],
                'password' => $this->service_data['password'],
            ]);

            // Store service data
            $service->setProvisionData($this->service_data);

            if ($result['success']) {
                $service->setProvisionStatus('completed');
                $this->logInfo('createJob', 'Service created successfully',
$result);
            } else {
                $service->setProvisionStatus('failed');
                $this->logError('createJob', 'Service creation failed', $result);
            }

            return ['status' => $result['success'] ? 'success' : 'error', 'data' =>
$result];

        } catch (Exception $e) {
            $this->logError('createJob', 'Exception in createJob', $e-
>getMessage());
            return ['status' => 'error', 'message' => $e->getMessage()];
        }
    }

    // Helper method for task queuing (based on real PUQcloud implementation)
```

```
private function queueModuleTask(string $method, array $tags = [], int $tries = 1): array
    {
        $data = [
            'module' => $this,                // Required: module instance
            'method' => $method,              // Required: method to execute
            'tries' => $tries,                // Optional: retry attempts
            'backoff' => 60,                  // Optional: delay between retries
            'timeout' => 600,                 // Optional: max execution time
            'maxExceptions' => 1,             // Optional: max exceptions
        ];

        Task::add('ModuleJob', 'Module', $data, $tags);
        return ['status' => 'success'];
    }

    // Admin Interface Configuration
    public function adminPermissions(): array
    {
        return [
            [
                'name' => 'View Servers',
                'key' => 'view-servers',
                'description' => 'View cloud servers',
            ],
            [
                'name' => 'Manage Servers',
                'key' => 'manage-servers',
                'description' => 'Create and manage cloud servers',
            ],
        ];
    }

    public function adminSidebar(): array
    {
        return [
            [
                'title' => 'Servers',
                'link' => 'servers',
                'active_links' => ['servers'],
                'permission' => 'view-servers',
```

```
        ],
    ];
}

public function adminWebRoutes(): array
{
    return [
        [
            'method' => 'get',
            'uri' => 'servers',
            'permission' => 'view-servers',
            'name' => 'servers',
            'controller' => 'MyCloudServiceController@servers',
        ],
    ];
}

// Client Area Configuration
public function getClientAreaMenuConfig(): array
{
    return [
        'general' => [
            'name' => 'General',
            'template' => 'client_area.general',
        ],
        'files' => [
            'name' => 'File Manager',
            'template' => 'client_area.files',
        ],
    ];
}

public function variables_general(): array
{
    return [
        'service_data' => $this->service_data,
        'config' => $this->config,
        'module_name' => $this->module_name,
        'service_uuid' => $this->service_uuid,
    ];
```

```
    }
}
```

Configuration File (config.php)

Standard structure based on real PUQcloud modules:

```
<?php

return [
    'name' => 'My Cloud Service',
    'description' => 'Professional cloud service module with ownPanel integration',
    'version' => '1.0.0',
    'author' => 'Your Name',
    'email' => 'your.email@domain.com',
    'website' => 'https://yourdomain.com',
    'logo' => __DIR__ . '/views/assets/img/logo.png',
    'icon' => 'fas fa-server',
];
```

Note: Store sensitive configuration in database models with encryption, not in `config.php`.

Database Model for Server Configuration:

```
// Models/CloudServer.php
class CloudServer extends Model
{
    protected $fillable = [
        'name', 'host', 'username', 'password', 'api_key', 'active'
    ];

    // Password automatically encrypted when saving
    public function setPasswordAttribute($value)
    {
        $this->attributes['password'] = Crypt::encryptString($value);
    }

    public function getPasswordAttribute($value)
    {
        return Crypt::decryptString($value);
```

```
    }
}
```

Access in module:

```
// Get server configuration from database
$server = CloudServer::where('active', true)->first();
$apiClient = new CloudAPIClient($server->host, $server->api_key);
```

Controller Example

```
<?php

namespace Modules\Product\MyCloudService\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use Illuminate\Http\JsonResponse;
use Illuminate\Contracts\View\View;

class MyCloudServiceController extends Controller
{
    public function servers(Request $request): View
    {
        $title = 'Cloud Servers';
        return view_admin_module('Product', 'MyCloudService', 'admin_area.servers',
compact('title'));
    }

    public function createServer(Request $request): JsonResponse
    {
        $validator = Validator::make($request->all(), [
            'name' => 'required|string|max:255',
            'ip' => 'required|ip',
            'location' => 'required|string',
        ]);

        if ($validator->fails()) {
            return response()->json([
                'success' => false,
```

```
                'errors' => $validator->errors()
            ], 422);
        }

        try {
            // Create server logic
            return response()->json([
                'success' => true,
                'message' => 'Server created successfully'
            ]);
        } catch (Exception $e) {
            return response()->json([
                'success' => false,
                'message' => $e->getMessage()
            ], 500);
        }
    }
}
```

Development Workflow

Phase 1: Planning and Design

1. **Define Requirements**
   - Service features and capabilities
   - External API integrations
   - Data models and relationships
   - User interface requirements
2. **Design Database Schema**

```
// In activate() method - Real PUQcloud pattern
Schema::create('cloud_server_groups', function (Blueprint $table) {
    $table->uuid()->primary();
    $table->string('name')->unique();
    $table->string('description')->nullable();
    $table->timestamps();
});

Schema::create('cloud_servers', function (Blueprint $table) {
    $table->uuid()->primary();
```

```
    $table->uuid('group_uuid');
    $table->string('name')->unique();
    $table->string('host');
    $table->string('username');
    $table->text('password'); // Encrypted using Crypt::encryptString()
    $table->text('api_key')->nullable(); // Encrypted API key
    $table->boolean('active')->default(true);
    $table->boolean('ssl')->default(true);
    $table->integer('port')->default(443);
    $table->integer('max_accounts')->default(0);
    $table->timestamps();

    $table->foreign('group_uuid')->references('uuid')->on('cloud_server_groups');
    $table->index(['active', 'group_uuid']);
});
```

Phase 2: Implementation

1. **Create Module Structure**
2. **Implement Core Methods**
3. **Add Controllers and Views**
4. **Configure Routes and Permissions**
5. **Add Language Files**

Phase 3: Testing

1. **Unit Tests**
2. **Integration Tests**
3. **Manual Testing**
4. **Performance Testing**

Phase 4: Deployment

1. **Install Module**
2. **Configure Settings**
3. **Activate Module**
4. **Verify Functionality**

Best Practices

Error Handling

```
public function provisionAccount(): array
{
    try {
        $this->validateConfiguration();
        $result = $this->callExternalAPI();
        $this->validateResponse($result);

        return ['success' => true, 'data' => $result];

    } catch (InvalidArgumentException $e) {
        $this->logError('provision', 'Configuration error', $e->getMessage());
        return ['success' => false, 'error' => 'Invalid configuration'];

    } catch (GuzzleException $e) {
        $this->logError('provision', 'API error', $e->getMessage());
        return ['success' => false, 'error' => 'External service unavailable'];

    } catch (Exception $e) {
        $this->logError('provision', 'Unexpected error', $e->getMessage());
        return ['success' => false, 'error' => 'Internal error occurred'];
    }
}
```

Security

```
public function saveServiceData(array $data = []): array
{
    // Sanitize input
    $data = array_map('trim', $data);

    // Validate with strict rules
    $validator = Validator::make($data, [
        'username' => 'required|alpha_dash|min:3|max:20',
        'password' => 'required|min:8|regex:/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/',
        'domain' => 'required|regex:/^[a-zA-Z0-9][a-zA-Z0-9-]*[a-zA-Z0-9]*\.[a-zA-
Z]{2,}$/',
    ]);
```

```
    // Encrypt sensitive data
    $data['password'] = encrypt($data['password']);

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

Performance

```
use Illuminate\Support\Facades\Cache;

public function getServerList(): array
{
    return Cache::remember('cloud_servers', 300, function () {
        return $this->apiClient->getServers();
    });
}
```

Logging

```
private function logOperation(string $operation, array $context = []): void
{
    $this->logInfo($operation, array_merge([
        'module' => $this->module_name,
        'service_uuid' => $this->service_uuid,
        'timestamp' => now()->toISOString(),
        'user_id' => auth()->id(),
    ], $context));
}
```

Advanced Features

Professional API Client Implementation

Based on real puqNextcloud API client pattern:

```
<?php
```

```
namespace Modules\Product\MyCloudService\Services;

use GuzzleHttp\Client;
use GuzzleHttp\Exception\GuzzleException;
use Illuminate\Support\Facades\Crypt;

class CloudAPIClient
{
    private Client $client;
    private string $apiKey;
    private string $baseUrl;
    private string $username;
    private string $password;

    public function __construct(array $serverConfig)
    {
        $this->baseUrl = "https://{$serverConfig['host']}:{$serverConfig['port']}";
        $this->apiKey = $serverConfig['api_key'] ?? '';
        $this->username = $serverConfig['username'];
        $this->password = Crypt::decryptString($serverConfig['password']);

        $this->client = new Client([
            'base_uri' => $this->baseUrl,
            'timeout' => 30,
            'connect_timeout' => 10,
            'headers' => [
                'Authorization' => 'Bearer ' . $this->apiKey,
                'Content-Type' => 'application/json',
                'User-Agent' => 'PUQcloud-Module/1.0',
                'Accept' => 'application/json',
            ],
        ]);
    }

    public function createAccount(array $data): array
    {
        return $this->makeRequest('POST', '/accounts', $data);
    }

    public function suspendAccount(string $username): array
```

```
    {
        return $this->makeRequest('PUT', "/accounts/{$username}/suspend");
    }

    public function deleteAccount(string $username): array
    {
        return $this->makeRequest('DELETE', "/accounts/{$username}");
    }

    private function makeRequest(string $method, string $endpoint, array $data = []):
array
    {
        try {
            $options = [];
            if (!empty($data)) {
                $options['json'] = $data;
            }

            $response = $this->client->request($method, $endpoint,
$options);

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

            return [
                'status' => 'success',
                'data' => $result,
                'http_code' => $response->getStatusCode(),
            ];

        } catch (GuzzleException $e) {
            return [
                'status' => 'error',
                'error' => $e->getMessage(),
                'http_code' => $e->getCode(),
            ];
        }
    }
}
```

Event Handling

```
use Illuminate\Support\Facades\Event;

public function createJob(): array
{
    Event::dispatch('cloud.account.creating', [
        'module' => $this->module_name,
        'service_uuid' => $this->service_uuid,
    ]);

    $result = $this->provisionAccount();

    Event::dispatch('hcloud.account.created', [
        'module' => $this->module_name,
        'service_uuid' => $this->service_uuid,
        'success' => $result['success'],
    ]);

    return $result;
}
```

Testing

Unit Tests

```
<?php

namespace Tests\Modules\Product\MyHostingService;

use Tests\TestCase;
use MyHostingService;

class MyHostingServiceTest extends TestCase
{
    private MyHostingService $module;
```

```
    protected function setUp(): void
    {
        parent::setUp();
        $this->module = new MyHostingService();
    }

    public function test_product_data_validation()
    {
        $data = [
            'server_location' => 'US-East',
            'plan_type' => 'premium',
            'disk_space' => 100,
            'bandwidth' => 1000,
        ];

        $result = $this->module->saveProductData($data);

        $this->assertEquals('success', $result['status']);
        $this->assertEquals(200, $result['code']);
    }

    public function test_invalid_product_data_rejected()
    {
        $data = [
            'server_location' => '',
            'plan_type' => 'invalid',
            'disk_space' => -1,
        ];

        $result = $this->module->saveProductData($data);

        $this->assertEquals('error', $result['status']);
        $this->assertEquals(422, $result['code']);
    }
}
```

Integration Tests

```
public function test_complete_service_lifecycle()
```

```
{
    // Create service
    $this->module->setServiceUuid('test-uuid');
    $result = $this->module->create();
    $this->assertEquals('success', $result['status']);

    // Suspend service
    $result = $this->module->suspend();
    $this->assertEquals('success', $result['status']);

    // Unsuspend service
    $result = $this->module->unsuspend();
    $this->assertEquals('success', $result['status']);

    // Terminate service
    $result = $this->module->termination();
    $this->assertEquals('success', $result['status']);
}
```

Troubleshooting

Common Issues

Module Not Loading

- Check file permissions (755 for directories, 644 for files)
- Verify class name matches filename
- Ensure proper namespace usage
- Check for PHP syntax errors

Routes Not Working

- Verify module is activated
- Check permission keys match
- Ensure controller exists and methods are public
- Clear route cache: `php artisan route:clear`

Jobs Not Processing

- Check queue configuration
- Verify Redis/database connection

- Run queue worker: `php artisan queue:work`
- Check failed jobs: `php artisan queue:failed`

Debugging Tools

```
// Add to any method for debugging
$this->logDebug('method_name', [
    'input' => $data,
    'service_uuid' => $this->service_uuid,
    'stack_trace' => debug_backtrace(DEBUG_BACKTRACE_IGNORE_ARGS, 5),
]);
```

Performance Monitoring

```
private function measurePerformance(callable $operation, string $operationName): mixed
{
    $startTime = microtime(true);
    $startMemory = memory_get_usage();

    $result = $operation();

    $endTime = microtime(true);
    $endMemory = memory_get_usage();

    $this->logInfo('performance', [
        'operation' => $operationName,
        'execution_time' => round(($endTime - $startTime) * 1000, 2) . 'ms',
        'memory_usage' => round(($endMemory - $startMemory) / 1024, 2) . 'KB',
        'peak_memory' => round(memory_get_peak_usage() / 1024 / 1024, 2) . 'MB',
    ]);

    return $result;
}
```

This comprehensive guide provides everything needed to develop professional modules for PUQcloud. Follow these patterns and best practices to ensure your modules are secure, performant, and maintainable.

API Reference

Base Module Class Methods

Core Methods

`__construct()`

Initializes the module instance. Always call `parent::__construct()` first.

```
public function __construct()
{
    parent::__construct();
    // Custom initialization logic here
}
```

Lifecycle Methods

`activate(): string`

Called when module is activated. Should set up database tables, initial configuration.

Returns: `'success'` or error message

```
public function activate(): string
{
    try {
        Schema::create('module_table', function (Blueprint $table) {
            $table->id();
            $table->uuid('service_uuid');
            $table->string('external_id')->nullable();
            $table->enum('status', ['active', 'suspended', 'terminated']);
            $table->timestamps();

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

```
        $this->logInfo('activate', 'Module activated successfully');
        return 'success';
    } catch (Exception $e) {
        $this->logError('activate', 'Activation failed: ' . $e->getMessage());
        return 'Error: ' . $e->getMessage();
    }
}
```

`deactivate(): string`

Called when module is deactivated. Should clean up resources.

Returns: `'success'` or error message

```
public function deactivate(): string
{
    try {
        Schema::dropIfExists('module_table');
        $this->logInfo('deactivate', 'Module deactivated successfully');
        return 'success';
    } catch (Exception $e) {
        $this->logError('deactivate', 'Deactivation failed: ' . $e->getMessage());
        return 'Error: ' . $e->getMessage();
    }
}
```

`update(): string`

Called when module version changes. Handle data migrations here.

Returns: `'success'` or error message

```
public function update(): string
{
    try {
        $currentVersion = $this->getCurrentVersion();

        if (version_compare($currentVersion, '1.1.0', '<')) {
            Schema::table('module_table', function (Blueprint $table) {
                $table->string('new_field')->nullable();
            });
        }
```

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

Product Module Methods

Configuration Methods

`getProductData(array $data = []): array`

Processes and stores product configuration data.

Parameters:

- `$data` - Array of configuration values

Returns: Processed configuration array

```
public function getProductData(array $data = []): array
{
    $this->product_data = [
        'server_location' => $data['server_location'] ?? '',
        'plan_type' => $data['plan_type'] ?? '',
        'disk_space' => $data['disk_space'] ?? '',
        'bandwidth' => $data['bandwidth'] ?? '',
    ];
    return $this->product_data;
}
```

`saveProductData(array $data = []): array`

Validates and saves product configuration.

Parameters:

- `$data` - Configuration data to validate

Returns:

```
```

```
[
    'status' => 'success|error',
    'message' => 'Error messages if validation fails',
    'code' => 200|422,
    'data' => $validatedData
]
```

Example:

```
public function saveProductData(array $data = []): array
{
    $validator = Validator::make($data, [
        'server_location' => 'required|string',
        'plan_type' => 'required|in:basic,premium,enterprise',
        'disk_space' => 'required|integer|min:1',
        'bandwidth' => 'required|integer|min:1',
    ]);

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

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

`getProductPage(): string`

Returns HTML for product configuration page in admin area.

Returns: Rendered HTML string

Notes: Should use `view()` to render Blade templates and pass validated configuration.

```

```

```
public function getProductPage(): string
{
    return $this->view('admin_area.product', [
        'config' => $this->product_data,
        'validation' => [/* rules, hints */],
    ]);
}
```

Service Methods

`getServiceData(array $data = []): array`

Processes service instance data.

Parameters:

- `$data` - Service configuration array

Returns: Processed service data

Example:

```
public function getServiceData(array $data = []): array
{
    $this->service_data = [
        'domain' => strtolower(trim($data['domain'] ?? '')),
        'username' => trim($data['username'] ?? ''),
        'notes' => $data['notes'] ?? null,
    ];
    return $this->service_data;
}
```

`saveServiceData(array $data = []): array`

Validates and saves service configuration.

Returns: Same format as `saveProductData()`

Example:

```
public function saveServiceData(array $data = []): array
{
    $validator = Validator::make($data, [
```

```
'domain' => 'required|regex:/^[a-zA-Z0-9][a-zA-Z0-9-]*[a-zA-Z0-9]*\.[a-zA-Z]{2,}$/',
        'username' => 'required|alpha_dash|min:3|max:20',
    ]);

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

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

`getServicePage(): string`

Returns HTML for service configuration page.

Returns: Rendered HTML string

Service Lifecycle Methods

`create(): array`

Initiates service creation process (usually queues a job).

Returns:

```
['status' => 'success|error', 'message' => 'Optional message']
```

Example:

```
public function create(): array
{
    $data = [
        'module' => $this,              // Module instance reference
        'method' => 'createJob',        // Method to execute
```

```
        'tries' => 1,                    // Retry attempts
        'backoff' => 60,                 // Delay between retries (seconds)
        'timeout' => 600,                // Max execution time
        'maxExceptions' => 1,            // Max exceptions before failure
    ];

    $service = Service::find($this->service_uuid);
    $service->setProvisionStatus('processing');
    Task::add('ModuleJob', 'Module', $data, ['create']);
    return ['status' => 'success'];
}
```

`createJob(): array`

Actual service creation logic executed asynchronously.

Returns: Status array

Example:

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

        // Your service creation logic here
        // ...

        $service->setProvisionStatus('completed');
        $this->logInfo('createJob', 'Service created successfully');
        return ['status' => 'success'];

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

`suspend(): array`

Initiates service suspension.

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

    Task::add('ModuleJob', 'Module', [
        'module' => $this,
        'method' => 'suspendJob',
        'tries' => 1,
        'backoff' => 60,
        'timeout' => 600,
        'maxExceptions' => 1,
    ], ['suspend', 'service:' . $this->service_uuid]);

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

`suspendJob(): array`

Actual suspension logic.

```
public function suspendJob(): array
{
    try {
        $api = new ExternalAPI($this->getServerConfig());
        $result = $api->suspend($this->service_data);

        if ($result['status'] === 'success') {
            Service::find($this->service_uuid)->setProvisionStatus('suspended');
            $this->logInfo('suspendJob', 'Service suspended');
            return ['status' => 'success'];
        }

        Service::find($this->service_uuid)->setProvisionStatus('failed');
        return ['status' => 'error', 'message' => $result['error'] ?? 'Unknown error'];
    } catch (Exception $e) {
        Service::find($this->service_uuid)->setProvisionStatus('failed');
        $this->logError('suspendJob', 'Suspension failed', $e->getMessage());
```

```
        return ['status' => 'error', 'message' => 'Suspension failed'];
    }
}
```

`unsuspend(): array`

Initiates service reactivation.

```
public function unsuspend(): array
{
    Task::add('ModuleJob', 'Module', [
        'module' => $this,
        'method' => 'unsuspendJob',
        'tries' => 1,
        'backoff' => 60,
        'timeout' => 600,
        'maxExceptions' => 1,
    ], ['unsuspend', 'service:' . $this->service_uuid]);

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

`unsuspendJob(): array`

Actual reactivation logic.

```
public function unsuspendJob(): array
{
    try {
        $api = new ExternalAPI($this->getServerConfig());
        $result = $api->unsuspend($this->service_data);
        if ($result['status'] === 'success') {
            Service::find($this->service_uuid)->setProvisionStatus('completed');
            return ['status' => 'success'];
        }
        return ['status' => 'error', 'message' => $result['error'] ?? 'Unknown error'];
    } catch (Exception $e) {
        $this->logError('unsuspendJob', 'Reactivation failed', $e->getMessage());
        return ['status' => 'error'];
    }
}
```

`termination(): array`

Initiates service termination.

```
public function termination(): array
{
    Task::add('ModuleJob', 'Module', [
        'module' => $this,
        'method' => 'terminationJob',
        'tries' => 1,
        'backoff' => 60,
        'timeout' => 600,
        'maxExceptions' => 1,
    ], ['terminate', 'service:' . $this->service_uuid]);
    return ['status' => 'success'];
}
```

`terminationJob(): array`

Actual termination logic.

```
public function terminationJob(): array
{
    try {
        $api = new ExternalAPI($this->getServerConfig());
```

```
        $result = $api->terminate($this->service_data);
        if ($result['status'] === 'success') {
            Service::find($this->service_uuid)->setProvisionStatus('terminated');
            return ['status' => 'success'];
        }
        return ['status' => 'error', 'message' => $result['error'] ?? 'Unknown error'];
    } catch (Exception $e) {
        $this->logError('terminationJob', 'Termination failed', $e->getMessage());
        return ['status' => 'error'];
    }
}
```

change_package(): array

Initiates package/plan change.

```
public function change_package(): array
{
    Task::add('ModuleJob', 'Module', [
        'module' => $this,
        'method' => 'change_packageJob',
        'tries' => 1,
        'backoff' => 60,
        'timeout' => 600,
        'maxExceptions' => 1,
    ], ['change_package', 'service:' . $this->service_uuid]);
    return ['status' => 'success'];
}
```

change_packageJob(): array

Actual package change logic.

```
public function change_packageJob(): array
{
    try {
        $api = new ExternalAPI($this->getServerConfig());
        $result = $api->changePackage([
            'plan' => $this->service_data['plan'] ?? 'basic',
            'bandwidth' => $this->product_data['bandwidth'] ?? null,
        ]);
```

```
        if ($result['status'] === 'success') {
            $this->logInfo('change_packageJob', 'Plan changed');
            return ['status' => 'success'];
        }
        return ['status' => 'error', 'message' => $result['error'] ?? 'Unknown error'];
    } catch (Exception $e) {
        $this->logError('change_packageJob', 'Change failed', $e->getMessage());
        return ['status' => 'error'];
    }
}
```

Plugin Module Methods

Lifecycle

`activate(): string` — create required tables, seed defaults. Must be idempotent.

`deactivate(): string` — drop or archive resources safely.

`update(): string` — migrate schema/config between versions.

Admin Interface

`adminSidebar(): array` — sidebar entries. See format in Admin Interface Methods.

`adminWebRoutes(): array` — web routes with permissions.

`adminApiRoutes(): array` — API routes for AJAX.

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

Background Work

Schedule jobs via `Task::add()` and listen to events in `hooks.php`.

```
Task::add('ModuleJob', 'Module', [
    'module' => $this,
    'method' => 'collectMetrics',
], ['metrics']);
```

Payment Module Methods

Configuration

`getModuleData(array $data = []): array` — normalize gateway settings.

Parameters: gateway keys, secrets, webhook secrets, sandbox flag.

Returns: sanitized config array.

Client UI

`getClientAreaHtml(array $data = []): string` — render payment form/session.

Parameters: `$data['invoice']`, optional customer/context.

Returns: rendered HTML.

```
public function getClientAreaHtml(array $data = []): string
```

```
{
    $invoice = $data['invoice'];
    $session = (new StripeClient($this->module_data))->createSession(
        referenceId: $invoice->uuid,
        invoiceId: $invoice->number,
        description: 'Invoice #' . $invoice->number,
        amount: $invoice->getDueAmountAttribute(),
        currency: $invoice->client->currency->code,
        return_url: $this->getReturnUrl(),
        cancel_url: $this->getCancelUrl(),
    );
    return $this->view('client_area', ['session' => $session]);
}
```

Settings Page

`getSettingsPage(array $data = []): string` — render admin configuration; should provide generated `webhook_url`.

Webhooks

`apiWebhookPost(Request $request): Response` — process gateway callbacks (POST).

`apiWebhookGet(Request $request): Response` — optional GET verification.

```
public function apiWebhookPost(Request $request): Response
{
    $payload = $request->all();
    // Verify signature and handle events
    switch ($payload['type'] ?? '') {
        case 'payment_intent.succeeded':
            return $this->onPaymentSuccess($payload);
        case 'payment_intent.payment_failed':
            return $this->onPaymentFailed($payload);
        default:
            return response('ok', 200);
    }
}
```

Payment Actions

`onPaymentSuccess(array $payload): Response|array` — mark invoice paid, log transaction.

`onPaymentFailed(array $payload): Response|array` — record failure.

`refund(string $transactionId, int|float $amount): array` — optional refund handler.

Notification Module Methods

Configuration

`getModuleData(array $data = []): array` — server, port, encryption, credentials, sender.

Delivery

`send(array $data = []): array` — send message through the channel.

Parameters:

- `to` (string) — recipient address
- `subject` (string) — subject/title
- `message` (string) — body (HTML or text)
- `attachments` (array) — optional attachments

Returns: `['status' => 'success']` or error with message/code.

```
public function send(array $data = []): array
{
    $validator = Validator::make($data, [
        'to' => 'required|email',
        'subject' => 'required|string|max:255',
        'message' => 'required|string',
    ]);
    if ($validator->fails()) {
        return ['status' => 'error', 'message' => $validator->errors(), 'code' => 422];
    }
    $mailer = $this->buildMailerConfiguration();
    Mail::mailer($mailer)->send(new NotificationMail(
        $data['to'], $data['subject'], $data['message'], $data['attachments'] ?? []
    ));
    return ['status' => 'success'];
}
```

Common Module Properties

The following properties are available to all module instances:

- `$module_name` (string) — module identifier
- `$module_type` (string) — one of: `Product`, `Plugin`, `Payment`, `Notification`
- `$config` (array) — values from `config.php`
- `$product_data` (array) — normalized product configuration (Product only)
- `$service_data` (array) — current service instance data (Product only)
- `$module_data` (array) — module-specific settings (Payment/Notification)
- `$service_uuid` (string) — current service UUID (Product)
- `$payment_gateway_uuid` (string) — payment gateway UUID (Payment)
- `$logger_context` (array) — default context for module logs

Webhook Endpoints

Modules can expose webhook endpoints via static routes. Typical signature:

```
// POST endpoint
public function apiWebhookPost(Request $request): Response
{
    // Validate, authenticate, process, respond
}

// GET endpoint
public function apiWebhookGet(Request $request): Response
{
    return response('ok', 200);
}
```

Admin Interface Methods

`adminPermissions(): array`

Defines module permissions for admin users.

Returns:

```
[
    [
```

```
        'name'  => 'Permission Display Name',
        'key'  => 'permission-key',
        'description'  => 'Permission description'
    ]
]
```

Example:

```
public function adminPermissions(): array
{
    return [
        [
            'name'  => 'View Servers',
            'key'  => 'view-servers',
            'description'  => 'View hosting servers',
        ],
        [
            'name'  => 'Manage Servers',
            'key'  => 'manage-servers',
            'description'  => 'Create and manage hosting servers',
        ],
    ];
}
```

`adminSidebar(): array`

Defines sidebar menu items in admin area.

Returns:

```
[
    [
        'title'  => 'Menu Title',
        'link'  => 'route-name',
        'active_links'  => ['route1', 'route2'],
        'permission'  => 'required-permission-key'
    ]
]
```

Link resolution:

- **link** must reference a valid route from `adminWebRoutes()`. It can be either the route `name`

or the `uri` defined there.
- The system resolves the item in this order: by route **name** → by route **uri** → fallback to a raw relative href.
- **Raw/fallback behavior**: if neither name nor uri matches a defined route, the menu item will render with a raw relative link (e.g., `/admin/modules/{Type}/{ModuleName}/{link}`). Such a link is considered broken and will lead to a non-working page (404). Keep links in sync with `adminWebRoutes()`.

Active state handling:

- `active_links` controls when the item (and its parent group) stays expanded and highlighted.
- The current admin route is matched against this array by route **name** or **uri**. A match sets the menu state to active/open.
- Use short keys or prefixes that you also use in `adminWebRoutes()` for consistent matching.

Working example:

```
public function adminWebRoutes(): array
{
    return [
        [
            'method' => 'get',
            'uri' => 'servers',
            'permission' => 'view-servers',
            'name' => 'servers',
            'controller' => 'ServerController@index',
        ],
        [
            'method' => 'get',
            'uri' => 'server-groups',
            'permission' => 'view-server-groups',
            'name' => 'server-groups',
            'controller' => 'ServerGroupController@index',
        ],
    ];
}

public function adminSidebar(): array
{
    return [
        [
```

```
            'title' => 'Server Management',
            'link' => 'servers', // resolves by route name
            'active_links' => ['servers', 'server-groups'],
            'permission' => 'manage-servers',
        ],
    ];
}
```

Broken link example (demonstration):

```
public function adminSidebar(): array
{
    return [
        [
            'title' => 'Diagnostics',
            'link' => 'diag', // NOT present in adminWebRoutes()
            'active_links' => ['diag'],
            'permission' => 'view-diagnostics',
        ],
    ];
}
// Result: menu renders an href like /admin/modules/Product/YourModule/diag
// Since no route exists, navigation will fail (404). Keep link and routes consistent.
```

`adminWebRoutes(): array`

Defines web routes for admin area.

Returns:

```
[
    [
        'method' => 'get|post|put|delete',
        'uri' => 'route/path',
        'permission' => 'required-permission',
        'name' => 'route.name',
        'controller' => 'ControllerName@methodName'
    ]
]
```

`adminApiRoutes(): array`

Defines API routes for admin area.

Returns: Same format as `adminWebRoutes()`

Client Area Methods

`getClientAreaMenuConfig(): array`

Defines client area menu tabs.

Returns:

```
[
    'tab_key' => [
        'name' => 'Tab Display Name',
        'template' => 'client_area.template_name'
    ]
]
```

Example:

```
public function getClientAreaMenuConfig(): array
{
    return [
        'general' => [
            'name' => 'General',
            'template' => 'client_area.general',
        ],
        'files' => [
            'name' => 'File Manager',
            'template' => 'client_area.files',
        ],
    ];
}
```

> **Note**: Each tab requires a corresponding `variables_{tab_name}()` method to provide data to the template.

`variables_{tab_name}(): array`

Provides variables for specific client area tab.

Returns: Array of variables for the template

Example:

```
public function variables_general(): array
{
    return [
        'service_data' => $this->service_data,
        'config' => $this->config,
        'status' => $this->getServiceStatus(),
    ];
}
```

`controllerClient_{tab_name}{Method}(Request $request): JsonResponse`

Handles AJAX requests from client area.

Parameters:

- `$request` - Laravel Request object

Returns: JsonResponse

Example:

```
public function controllerClient_generalGet(Request $request): JsonResponse
{
    try {
        $data = $this->getServiceDetails();
        return response()->json([
            'success' => true,
            'data' => $data,
        ]);
    } catch (Exception $e) {
        return response()->json([
            'success' => false,
            'message' => $e->getMessage(),
        ], 500);
    }
}
```

Utility Methods

`view(string $template, array $data = []): string`

Renders module template.

Parameters:

- `$template` - Template path relative to module views directory
- `$data` - Variables to pass to template

Returns: Rendered HTML

Example:

```
public function getServicePage(): string
{
    return $this->view('admin_area.service', [
        'service_data' => $this->service_data,
        'config' => $this->config,
    ]);
}
```

`config(string $key): mixed`

Gets configuration value from `config.php`.

Parameters:

- `$key` - Configuration key

Returns: Configuration value

Example:

```
$apiUrl = $this->config('api_url');
$apiKey = $this->config('api_key');
```

Logging Methods

`logInfo(string $action, array|string $request = [], array|string $response = []): void`

Logs informational message.

Example:

```
$this->logInfo('service_created', [
    'service_uuid' => $this->service_uuid,
    'product_data' => $this->product_data,
], ['status' => 'success']);
```

logError(string $action, array|string $request = [], array|string $response = []): void

Logs error message.

Example:

```
$this->logError('api_call_failed', [
    'endpoint' => '/api/create',
    'data' => $requestData,
], ['error' => $e->getMessage()]);
```

logDebug(string $action, mixed $request = [], mixed $response = []): void

Logs debug message.

Example:

```
$this->logDebug('processing_step', [
    'step' => 'validation',
    'data' => $inputData,
]);
```

Task System

Task::add()

Queues a background job.

```
Task::add($jobName, $queue, $inputData, $tags);
```

Parameters:

- `$jobName` - Job class name (e.g., 'ModuleJob')
- `$queue` - Queue name (e.g., 'Module')
- `$inputData` - Array of data to pass to job
- `$tags` - Array of tags for job identification

Job Data Structure for ModuleJob:

```
$data = [
    'module' => $this,                    // Module instance (required)
    'method' => 'methodToCall',           // Method name to execute (required)
    'tries' => 1,                         // Number of retry attempts (default: 1)
    'backoff' => 60,                      // Delay between retries in seconds (default: 10)
    'timeout' => 600,                     // Maximum execution time in seconds (default:
600)
    'maxExceptions' => 1,                 // Max exceptions before failure (default: 2,
recommended: 1)
];

// Tags for job identification and filtering
$tags = ['create', 'service:' . $this->service_uuid];

Task::add('ModuleJob', 'Module', $data, $tags);
```

Complete Example:

```
public function create(): array
{
    $data = [
        'module' => $this,
        'method' => 'createJob',
        'tries' => 1,
        'backoff' => 60,
        'timeout' => 600,
        'maxExceptions' => 1,
    ];

    $tags = [
```

```
        'create',
        'hosting',
        'service:' . $this->service_uuid,
    ];

    $service = Service::find($this->service_uuid);
    $service->setProvisionStatus('processing');
    Task::add('ModuleJob', 'Module', $data, $tags);
    return ['status' => 'success'];
}
```

Helper Functions

view_admin_module()

Renders admin module view.

```
view_admin_module($type, $name, $view, $data = [], $mergeData = [])
```

Parameters:

- `$type` - Module type (e.g., 'Product')
- `$name` - Module name
- `$view` - View path
- `$data` - View data
- `$mergeData` - Additional data to merge

Example:

```
public function dashboard(Request $request): View
{
    $title = 'Server Dashboard';
    return view_admin_module('Product', 'MyHostingService', 'admin_area.dashboard',
compact('title'));
}
```

logModule()

Direct logging function.

```
logModule($type, $name, $action, $level, $request = [], $response = [])
```

Parameters:

- `$type` - Module type
- `$name` - Module name
- `$action` - Action being performed
- `$level` - Log level ('info', 'error', 'debug')
- `$request` - Request data
- `$response` - Response data

Service Model Methods

setProvisionStatus()

Updates service provisioning status.

```
$service->setProvisionStatus($status);
```

Status Values:

- `'pending'` - Waiting to be processed
- `'processing'` - Currently being processed
- `'completed'` - Successfully completed (active service)
- `'failed'` - Failed processing
- `'error'` - Error occurred during processing
- `'suspended'` - Service is suspended
- `'pause'` - Service is paused/idle
- `'terminated'` - Service is terminated

Example:

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

        // Generate service credentials
        $this->service_data['username'] = $this->product_data['username_prefix']
.
```

```
random_int(100000, 999999) .
                                        $this->product_data['username_suffix'];
        $this->service_data['password'] = generateStrongPassword(10);

        // Create API client and provision account
        $apiClient = new HostingAPIClient($this->config('api_url'), $this-
>config('api_key'));
        $result = $apiClient->createAccount([
            'domain'  => $this->service_data['domain'],
            'username'  => $this->service_data['username'],
            'password'  => $this->service_data['password'],
        ]);

        if ($result['status'] === 'success') {
            $service->setProvisionData($this->service_data);
            $service->setProvisionStatus('completed');
            $this->logInfo('createJob', 'Service created successfully');
            return ['status'  => 'success'];
        } else {
            $service->setProvisionStatus('failed');
            $this->logError('createJob', 'Service creation failed', $result);
            return ['status'  => 'error', 'message'  => $result['error']];
        }

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

Validation Rules

Common Validation Patterns

```
// Domain validation
'domain'  => 'required|regex:/^[a-zA-Z0-9][a-zA-Z0-9-]*[a-zA-Z0-9]*\.[a-zA-Z]{2,}$/',
```

```
// Username validation
'username' => 'required|alpha_dash|min:3|max:20',

// Strong password
'password' => 'required|min:8|regex:/^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)/',

// IP address
'ip' => 'nullable|ip',

// Plan/package validation
'plan' => 'required|in:basic,premium,enterprise',

// Disk space (in GB)
'disk_space' => 'required|integer|min:1|max:1000',
```

Error Handling Patterns

Standard Error Response

```
return [
    'status' => 'error',
    'message' => 'User-friendly error message',
    'errors' => $validator->errors(), // For validation errors
    'code' => 422, // HTTP status code
];
```

Exception Handling in Jobs

```
public function createJob(): array
{
    try {
        // Job logic here
        return ['status' => 'success'];

    } catch (ExternalServiceException $e) {
        $this->logError('createJob', 'External service error', $e->getMessage());
        return ['status' => 'error', 'message' => 'Service temporarily
unavailable'];
```

```
    } catch (ValidationException $e) {
        $this->logError('createJob', 'Validation error', $e->errors());
        return ['status' => 'error', 'message' => 'Invalid data provided'];

    } catch (Exception $e) {
        $this->logError('createJob', 'Unexpected error', $e->getMessage());
        return ['status' => 'error', 'message' => 'An unexpected error occurred'];
    }
}
```

Performance Best Practices

Caching

```
use Illuminate\Support\Facades\Cache;

// Cache expensive operations
$servers = Cache::remember("module_{$this->module_name}_servers", 300, function() {
    return $this->fetchServersFromAPI();
});

// Cache with tags for easier invalidation
Cache::tags(['module', $this->module_name])->put('key', $value, 300);

// Invalidate cache
Cache::tags(['module', $this->module_name])->flush();
```

Database Optimization

```
// Use database transactions for multiple operations
DB::transaction(function() use ($data) {
    $this->createUser($data);
    $this->assignPermissions($data);
    $this->sendWelcomeEmail($data);
});

// Eager loading relationships
```

```
$services = Service::with(['product', 'customer'])->get();
```

Memory Management

```
// Process large datasets in chunks
Service::chunk(100, function($services) {
    foreach ($services as $service) {
        $this->processService($service);
    }
});
```

Security Guidelines

Input Sanitization

```
// Always sanitize input
$data = array_map('trim', $data);
$data = array_map('strip_tags', $data);

// For HTML content, use proper escaping
$safeHtml = htmlspecialchars($userInput, ENT_QUOTES, 'UTF-8');
```

Sensitive Data Handling

```
// Encrypt sensitive data before storage
$encryptedPassword = encrypt($password);

// Don't log sensitive information
$this->logInfo('user_created', [
    'username' => $username,
    'email' => $email,
    // Don't log password or API keys
]);
```

Permission Checking

```
// Always check permissions in controllers
if (!$admin->hasPermission('Product-ModuleName-action')) {
```

```
    abort(403, 'Insufficient permissions');
}
```

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
<?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.