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
<?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.

Revision #7
Created 30 July 2025 09:25:41 by Dmytro Kravchenko
Updated 13 August 2025 12:11:33 by Dmytro Kravchenko