PUQcloud Module Development Documentation

COMPREHENSIVE GUIDE: Complete documentation for developing professional modules in PUQcloud billing system.

Documentation Structure

This documentation suite provides everything you need to build professional, scalable modules for PUQcloud:

Start Here

- **What is a Module?** - Complete introduction to PUQcloud's modular architecture

Development Resources

- **Module Development Guide** - Step-by-step development tutorial with real examples
- **API Reference** - Complete method documentation and usage examples
- **Practical Examples** - Real-world module implementations

Support & Quality

- **FAQ & Troubleshooting** - Solutions to common development problems
- **Development Checklist** - Quality assurance and best practices guide

PUQcloud Module Architecture

Module System Overview

PUQcloud uses a sophisticated modular architecture with four distinct module types:

```
App\Modules\Module (Base Class)
├── Product      → Service lifecycle management (hosting, VPS, domains)
├── Plugin       → System extensions (monitoring, analytics, reports)
├── Payment      → Payment gateway integrations (Stripe, PayPal, crypto)
└── Notification → Communication channels (SMTP, SMS, Slack, webhooks)
```

Class Responsibilities and APIs

Every module is a PHP class inheriting from `App\Modules\Module` and shares a consistent contract:

- **Lifecycle**: `activate()`, `update()`, `deactivate()`
- **Rendering**: `view(string $template, array $data = [])`
- **Configuration**: `config(string $key): mixed`
- **Async jobs**: `Task::add('ModuleJob', 'Module', $data, $tags)`
- **Logging**: `logInfo()`, `logError()`, `logDebug()`
- **Security**: permissions and access control in admin routes/controllers

Derived Module Types and Typical Methods

Product (Service Management)

- **Product config**: `getProductData()`, `saveProductData()`, `getProductPage()`
- **Service config**: `getServiceData()`, `saveServiceData()`, `getServicePage()`
- **Service lifecycle**: `create()`/`createJob()`, `suspend()`/`suspendJob()`, `unsuspend()`/`unsuspendJob()`, `termination()`/`terminationJob()`, `change_package()`/`change_packageJob()`
- **Client area**: `getClientAreaMenuConfig()`, `variables_{tab}()`, `controllerClient_{tab}{Method}()`
- **Admin area**: `adminPermissions()`, `adminSidebar()`, `adminWebRoutes()`, `adminApiRoutes()`

Plugin (System Extensions)

- **Lifecycle**: `activate()`, `deactivate()`, `update()`
- **Admin**: `adminSidebar()`, `adminWebRoutes()`, `adminApiRoutes()`
- **Background work**: scheduled tasks via `Task::add()` and event hooks in `hooks.php`
- **Client area**: typically none

Payment (Payment Processing)

- **Config**: `getModuleData()`, admin settings via `getSettingsPage()`
- **Checkout UI**: `getClientAreaHtml()`

- **Webhooks**: `apiWebhookPost()`/`apiWebhookGet()`
- **Actions**: success/failure handlers, optional `refund()`

Notification (Communication Channels)

- **Config**: `getModuleData()` (server, credentials, encryption, sender)
- **Delivery**: `send(array $data)` with validation, retries, logging
- **Templates**: render content with Blade templates

Real Module Structure

Every module follows this standardized structure:

```
modules/{Type}/{ModuleName}/
├── {ModuleName}.php              # Main module class (required)
├── config.php                    # Module metadata (required)
├── hooks.php                     # Event hooks (optional)
├── Controllers/                  # HTTP request handlers
│   └── {ModuleName}Controller.php
├── Services/                     # External API clients
│   └── {ExternalService}Client.php
├── Models/                       # Database models
│   └── {ModuleName}Model.php
├── views/                        # Blade templates
│   ├── admin_area/              # Admin interface views
│   ├── client_area/             # Client interface views
│   └── assets/                  # CSS, JS, images
└── lang/                         # Internationalization
    ├── en.php
    └── pl.php
```

□□ Quick Start Examples

Product Module (Service Management)

Real implementation example based on puqNextcloud:

```
<?php

use App\Models\Service;
use App\Models\Task;
```

```php
use App\Modules\Product;

class MyNextcloudService extends Product
{
    public function __construct()
    {
        parent:: __construct();
    }

    // Product configuration
    public function getProductData(array $data = []): array
    {
        $this->product_data = [
            'group_uuid' => $data['group_uuid'] ?? '',
            'username_prefix' => $data['username_prefix'] ?? 'nc_',
            'username_suffix' => $data['username_suffix'] ?? '',
            'quota' => $data['quota'] ?? '1GB',
        ];
        return $this->product_data;
    }

    // Service creation (asynchronous)
    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
        ];

        $tags = ['create'];

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

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

```
    }

    // Actual provisioning logic
    public function createJob(): array
    {
        try {
            $service = Service::find($this->service_uuid);

            // 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
            $apiClient = new NextcloudAPIClient($this-
>getServerConfig());

            // Provision account via API
            $response = $apiClient->createUser([
                'userid'  => $this->service_data['username'],
                'password'  => $this->service_data['password'],
            ]);

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

        } catch (Exception $e) {
```

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

Payment Module (Stripe Integration)

Real implementation example based on puqStripe:

```
<?php

use App\Modules\Payment;

class MyStripePayment extends Payment
{
    public function getModuleData(array $data = []): array
    {
        return [
            'publishable_key' => $data['publishable_key'] ?? '',
            'secret_key' => $data['secret_key'] ?? '',
            'webhook_secret' => $data['webhook_secret'] ?? '',
            'sandbox' => $data['sandbox'] ?? false,
        ];
    }

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

        $stripe = new StripeClient($this->module_data);

        $session = $stripe->createSession(
            referenceId: $invoice->uuid,
            invoiceId: $invoice->number,
            description: 'Invoice #' . $invoice->number,
            amount: $amount,
```

```
            currency: $currency,
            return_url: $this->getReturnUrl(),
            cancel_url: $this->getCancelUrl(),
        );

        return $this->view('client_area', ['session' => $session]);
    }
}
```

Notification Module (SMTP Email)

Real implementation example based on puqSMTP:

```
<?php

use App\Modules\Notification;

class MySMTPNotification extends Notification
{
    public function getModuleData(array $data = []): array
    {
        return [
            'email' => $data['email'] ?? '',
            'server' => $data['server'] ?? '',
            'sender_name' => $data['sender_name'] ?? '',
            'port' => $data['port'] ?? 587,
            'encryption' => $data['encryption'] ?? 'tls',
            'username' => $data['username'] ?? '',
            'password' => $data['password'] ?? '',
        ];
    }

    public function send(array $data = []): array
    {
        try {
            $config = $this->buildMailConfig();

            Mail::mailer($config)->send(new NotificationMail(
                $data['to'],
                $data['subject'],
```

```
                $data['message']
            ));

            $this->logInfo('send', 'Email sent successfully', [
                'to' => $data['to'],
                'subject' => $data['subject']
            ]);

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

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

□□ Key Implementation Details

Task System (Asynchronous Processing)

Correct Task::add usage:

```
// Standard task configuration
$data = [
    'module' => $this,                    // Required: module instance
    'method' => 'methodName',             // Required: method to execute
    'tries' => 1,                         // Optional: retry attempts (default: 1)
    'backoff' => 60,                      // Optional: delay between retries
    'timeout' => 600,                     // Optional: max execution time
    'maxExceptions' => 1,                 // Optional: max exceptions
];

$tags = ['operation_type', 'service:' . $this->service_uuid];
Task::add('ModuleJob', 'Module', $data, $tags);
```

Database Integration

Table creation in activate():

```
```

```
public function activate(): string
{
    try {
        if (!Schema::hasTable('module_servers')) {
            Schema::create('module_servers', function (Blueprint $table) {
                $table->uuid()->primary();
                $table->string('name')->unique();
                $table->string('host');
                $table->string('username');
                $table->string('password');
                $table->boolean('active')->default(true);
                $table->integer('port')->default(443);
                $table->timestamps();
            });
        }

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

Configuration File

Standard config.php structure:

```php
<?php

return [
    'name' => 'My Module Name',
    'description' => 'Professional module description',
    'version' => '1.0.0',
    'author' => 'Developer Name',
    'email' => 'developer@domain.com',
    'website' => 'https://developer-website.com',
    'logo' => __DIR__ . '/views/assets/img/logo.png',
    'icon' => 'fas fa-server',  // FontAwesome icon
];
```

□□ Module Types Comparison

Feature	Product	Plugin	Payment	Notification
Purpose	Service lifecycle	System extension	Payment processing	Communication
Base Class	`App\Modules\Product`	`App\Modules\Plugin`	`App\Modules\Payment`	`App\Modules\Notificatio`
Key Methods	`create()`, `suspend()`, `terminate()`	`activate()`, custom methods	`getClientAreaHtml()`	`send()`
Client Area	Full interface with tabs	None	Payment forms	Configuration only
Examples	Nextcloud, VPS hosting	Monitoring, analytics	Stripe, PayPal, MonoBank	SMTP, SMS

□□ Development Environment

Prerequisites

```
# Required stack
PHP 8.1+
Laravel 9.0+
Composer
MySQL/PostgreSQL
```

```
Redis (for queues)
Node.js & NPM (for assets)

# Recommended tools
Docker & Docker Compose
Git
PHPStorm/VSCode
```

Quick Setup

```
# 1. Create module directory
mkdir -p modules/Product/MyModule

# 2. Create basic files
cat > modules/Product/MyModule/MyModule.php << 'EOF'
<?php

use App\Modules\Product;

class MyModule extends Product
{
    public function __construct()
    {
        parent::__construct();
    }
}
EOF

# 3. Create configuration
cat > modules/Product/MyModule/config.php << 'EOF'
<?php

return [
    'name' => 'My Module',
    'description' => 'Module description',
    'version' => '1.0.0',
    'author' => 'Your Name',
    'email' => 'your.email@domain.com',
    'website' => 'https://yourdomain.com',
    'icon' => 'fas fa-server',
```

```
];
EOF
```

□□ Next Steps

Learning Path

- □□ **Read** **What is a Module?** - Understand the architecture
- □□ **Follow** **Development Guide** - Build your first module
- □□ **Use** **API Reference** - Implement specific features
- □□ **Study** **Examples** - Learn from real implementations
- □ **Apply** **Checklist** - Ensure quality

Development Support

Issue Type	Resource	Description
Getting Started	Development Guide	Step-by-step tutorial
Technical Issues	FAQ & Troubleshooting	Common problems & solutions
API Questions	API Reference	Method documentation
Best Practices	Development Checklist	Quality guidelines

⚡ Key Success Factors

□□ Accuracy First

- Follow real implementation patterns from existing modules
- Use correct `Task::add()` syntax and parameters
- Implement proper error handling and logging

□□ Security & Performance

- Validate all inputs with Laravel Validator
- Use encrypted storage for sensitive data
- Implement caching for expensive operations
- Handle external API failures gracefully

Quality Assurance

- Test all module functionality thoroughly
- Follow PSR-12 coding standards
- Use proper type hints and documentation
- Implement comprehensive error handling

Ready to build professional modules? Start with **What is a Module?** and create modules that integrate seamlessly with PUQcloud's architecture!

Revision #6
Created 30 July 2025 01:09:51 by Dmytro Kravchenko
Updated 13 August 2025 05:11:33 by Dmytro Kravchenko