Certificate Service

CertificateService Class

Anonymous · 09/09/2026

CertificateService Class

The `CertificateService` is the main entry point for working with digital certificates in the Derafu Certificate library. It provides a unified interface to all the key functionality like loading, validating, and creating certificates.

Overview

The `CertificateService` follows the service pattern, acting as a facade for:

- `CertificateFaker`: For creating test certificates.
- `CertificateLoader`: For loading certificates from various sources.
- `CertificateValidator`: For validating certificates.

By using the service, you can access all the library's functionality through a single interface.

Basic Usage

Setting Up the Service

```
use Derafu\Certificate\Service\CertificateLoader;
use Derafu\Certificate\Service\CertificateFaker;
use Derafu\Certificate\Service\CertificateValidator;
use Derafu\Certificate\Service\CertificateService;

// Create dependencies.
$loader = new CertificateLoader();
$faker = new CertificateFaker($loader);
$validator = new CertificateValidator();

// Create the service.
$service = new CertificateService($faker, $loader, $validator);
```

Loading Certificates

The service provides methods to load certificates from various sources:

```
// Load from a PKCS#12 file.
$certificate = $service->loadFromFile('/path/to/certificate.p12', 'password');

// Load from PKCS#12 data.
$certificate = $service->loadFromData($pkcs12Data, 'password');

// Load from a certificate array.
$certificate = $service->loadFromArray([
    'cert' => $publicKey,
    'pkey' => $privateKey
]);

// Load directly from key strings.
$certificate = $service->loadFromKeys($publicKey, $privateKey);
```

Validating Certificates

```
use Derafu\Certificate\Exception\CertificateException;

try {
    $service->validate($certificate);
    echo "Certificate is valid!";
} catch (CertificateException $e) {
    echo "Validation failed: " . $e->getMessage();
}
```

Creating Test Certificates

```
// Create with default values.
$certificate = $service->createFake();

// Create with custom values.
$certificate = $service->createFake(
    id: '12345678-9',
    name: 'John Doe',
    email: 'john.doe@example.com'
);

// Use the certificate.
$id = $certificate->getId();
$name = $certificate->getName();
$isValid = $certificate->isActive();
```

Complete Example

```
// Initialize the service.
$loader = new CertificateLoader();
$faker = new CertificateFaker($loader);
$validator = new CertificateValidator();
$service = new CertificateService($faker, $loader, $validator);

// Create a test certificate.
$certificate = $service->createFake(
    id: '12345678-9',
    name: 'John Doe',
    email: 'john.doe@example.com'
);

// Extract certificate data.
echo "Certificate ID: " . $certificate->getId() . "\n";
echo "Certificate Name: " . $certificate->getName() . "\n";
echo "Valid until: " . $certificate->getTo() . "\n";
echo "Days remaining: " . $certificate->getExpirationDays() . "\n";

// Validate the certificate.
try {
    $service->validate($certificate);
    echo "Certificate is valid\n";
} catch (CertificateException $e) {
    echo "Invalid certificate: " . $e->getMessage() . "\n";
}

// Export the certificate.
$pkcs12Data = $certificate->getPkcs12('password');
file_put_contents('certificate.p12', $pkcs12Data);

// Later, load the certificate back.
$loadedCertificate = $service->loadFromFile('certificate.p12', 'password');
```

Customizing the Service

Custom Validator

You can create a custom validator by implementing the `CertificateValidatorInterface`:

```
use Derafu\Certificate\Contract\CertificateInterface;
use Derafu\Certificate\Contract\CertificateValidatorInterface;
use Derafu\Certificate\Exception\CertificateException;

class MyCustomValidator implements CertificateValidatorInterface
```

```
{
    public function validate(CertificateInterface $certificate): void
    {
        // Implement your validation logic.
        if (!$certificate->isActive()) {
            throw new CertificateException(
                'Certificate is expired.'
            );
        }

        // Add specific validation for your use case.
        if ($certificate->getExpirationDays() < 30) {
            throw new CertificateException(
                'Certificate will expire in less than 30 days.'
            );
        }
    }
}

// Use your custom validator.
$customValidator = new MyCustomValidator();
$service = new CertificateService($faker, $loader, $customValidator);
```

Service Components

CertificateLoader

The `CertificateLoader` component handles loading certificates from various sources:

- PKCS#12 files.
- PKCS#12 binary data.
- Array of keys.
- Individual public and private keys.

CertificateFaker

The `CertificateFaker` component provides a way to create self-signed certificates for testing purposes:

- Create certificates with custom subject information.
- Create certificates with custom issuer information.
- Create certificates with custom validity periods.

Under the hood, it uses the `SelfSignedCertificate` class.

CertificateValidator

The `CertificateValidator` component validates certificates according to specific requirements:

- Checks if the certificate ID (RUN) is properly formatted.
- Ensures that “K” in the ID is uppercase.
- Verifies that the certificate is not expired.

The default validator is designed for Chilean certificates used with the SII (Servicio de Impuestos Internos), but you can implement your own validator for different requirements.

Last updated on 09/09/2026