Architecture

Derafu Mail Architecture

Anonymous · 09/09/2026

Derafu Mail Architecture

This guide explains the architecture of the Derafu Mail library, which is built on the Derafu Backbone architectural framework. Understanding this architecture will help you effectively use and extend the library.

Architectural Overview

Derafu Mail follows a hierarchical architecture that separates concerns into distinct layers, each with specific responsibilities:

1. **Package Layer**: The entry point and container for all components.
2. **Component Layer**: Functional modules within the package.
3. **Worker Layer**: Task executors within components.
4. **Handler Layer**: Process orchestrators that coordinate strategies.
5. **Strategy Layer**: Specific implementations of mail operations.
6. **Model Layer**: Domain models representing email-related entities.

Package Layer

The `MailPackage` serves as the entry point for the entire library, following the Package pattern from Derafu Backbone.

```
#[Package(name: 'mail')]
class MailPackage extends AbstractPackage implements MailPackageInterface
```

Responsibilities:

- Provides access to components (currently just `ExchangeComponent`).
- Acts as the root of the dependency tree.
- Maintains registry information for service discovery.

Component Layer

The `ExchangeComponent` represents a specific functional area within the mail domain, handling both sending and receiving of emails.

```
#[Component(name: 'exchange', package: 'mail')]
class ExchangeComponent extends AbstractComponent implements
ExchangeComponentInterface
```

Responsibilities:

- Manages workers for sending and receiving emails.
- Provides access to these workers through getter methods.
- Groups related functionality under a single namespace.

Worker Layer

Workers expose the public API for specific tasks:

SenderWorker

```
#[Worker(name: 'sender', component: 'exchange', package: 'mail')]
class SenderWorker extends AbstractWorker implements SenderWorkerInterface
```

Responsibilities:

- Provides a public `send()` method for sending emails.
- Delegates the actual sending process to handlers.
- Manages any worker-specific resources.

ReceiverWorker

```
#[Worker(name: 'receiver', component: 'exchange', package: 'mail')]
class ReceiverWorker extends AbstractWorker implements ReceiverWorkerInterface
```

Responsibilities:

- Provides a public `receive()` method for receiving emails.
- Delegates the receiving process to handlers.
- Manages any worker-specific resources.

Handler Layer

Handlers orchestrate complex processes, selecting appropriate strategies and managing the workflow:

SendHandler

```
class SendHandler extends AbstractHandler
```

Responsibilities:

- Selects the appropriate sending strategy based on configuration.
- Orchestrates the email sending process.
- Handles errors in a centralized manner.
- Manages options and configuration.

ReceiveHandler

```
class ReceiveHandler extends AbstractHandler
```

Responsibilities:

- Selects the appropriate receiving strategy based on configuration.
- Orchestrates the email receiving process.
- Handles errors in a centralized manner.
- Manages options and configuration.

Strategy Layer

Strategies implement specific methods for sending or receiving emails:

SmtpStrategy

```
#[Strategy(name: 'smtp', worker: 'sender', component: 'exchange', package: 'mail')]
class SmtpStrategy extends AbstractMailerStrategy implements SenderStrategyInterface
```

Responsibilities:

- Configures and uses Symfony Mailer for SMTP transport.
- Builds appropriate DSN string based on configuration.
- Handles the actual sending of emails.
- Manages SMTP-specific settings.

ImapStrategy

```
#[Strategy(name: 'imap', worker: 'receiver', component: 'exchange', package: 'mail')]
class ImapStrategy extends AbstractMailboxStrategy implements
ReceiverStrategyInterface
```

Responsibilities:

- Configures and uses PHP-IMAP for IMAP access.
- Builds appropriate DSN string for IMAP connection.
- Searches for and retrieves emails based on criteria.
- Manages IMAP-specific settings.

Abstract Base Classes

The library provides abstract base classes that implement common functionality:

AbstractMailerStrategy

```
abstract class AbstractMailerStrategy extends AbstractStrategy implements
SenderStrategyInterface
```

Responsibilities:

- Provides common code for email sending strategies.
- Handles the creation of Symfony Mailer instances.
- Processes envelopes and messages.

AbstractMailboxStrategy

```
abstract class AbstractMailboxStrategy extends AbstractStrategy implements
ReceiverStrategyInterface
```

Responsibilities:

- Provides common code for email receiving strategies.
- Handles the creation of Mailbox instances.
- Processes received emails into envelopes.

Model Layer

Domain models represent the core entities of the email domain:

Envelope

```
class Envelope extends SymfonyEnvelope implements EnvelopeInterface
```

Responsibilities:

- Contains sender and recipient information.
- Holds one or more messages.
- Provides a container for email communication.

Message

```
class Message extends SymfonyEmail implements MessageInterface
```

Responsibilities:

- Represents an individual email message.
- Contains subject, body, attachments, etc.
- Tracks sending/receiving errors.

Postman

```
class Postman implements PostmanInterface
```

Responsibilities:

- Acts as a transport container for envelopes.
- Holds configuration options for sending/receiving.
- Provides a unified interface for email operations.

Mailbox

```
class Mailbox implements MailboxInterface
```

Responsibilities:

- Represents an email mailbox (IMAP folder).
- Provides methods for searching and retrieving emails.
- Handles connection to mail servers.

Data Flow Examples

Sending an Email

1. Client code creates a Message and adds it to an Envelope.
2. Envelope is added to a Postman with SMTP configuration.
3. Client calls SenderWorker's send() method with the Postman.
4. SenderWorker delegates to SendHandler.
5. SendHandler selects SmtpStrategy based on configuration.
6. SmtpStrategy uses Symfony Mailer to send the emails.
7. Results are returned through the same chain.

Receiving Emails

1. Client creates a Postman with IMAP configuration.
2. Client calls ReceiverWorker's receive() method with the Postman.
3. ReceiverWorker delegates to ReceiveHandler.
4. ReceiveHandler selects ImapStrategy based on configuration.
5. ImapStrategy connects to the mailbox and retrieves emails.
6. Retrieved emails are converted to Envelopes with Messages.
7. Envelopes are added to the Postman and returned.

Extending the Library

The architecture makes it easy to extend the library with new strategies:

1. **New Sending Strategy**: Create a class that extends AbstractMailerStrategy or implements SenderStrategyInterface.
2. **New Receiving Strategy**: Create a class that extends AbstractMailboxStrategy or implements ReceiverStrategyInterface.
3. **Tag with Attribute**: Use the #[Strategy] attribute for automatic discovery.
4. **Use via Configuration**: Specify your strategy in the Postman options.

Key Benefits of This Architecture

1. **Separation of Concerns**: Each class has a single, well-defined responsibility.
2. **Extensibility**: Easy to add new strategies without modifying existing code.
3. **Testability**: Clean interfaces make unit testing straightforward.
4. **Configurability**: Comprehensive options at every level.
5. **Maintainability**: Clear structure makes code easy to understand and modify.

Conclusion

Derafu Mail's architecture provides a solid foundation for email operations in PHP applications. By leveraging the Derafu Backbone patterns, it achieves a clean separation of concerns while remaining

flexible and extensible.

Understanding this architecture will help you effectively use the library and extend it with new capabilities when needed.

Last updated on 09/09/2026