The Definitive Guide to Yii 2.0

Download ▾ English ▾ Version 2.0 ▾

Search the

INTRODUCTION

GETTING STARTED

APPLICATION STRUCTURE

HANDLING REQUESTS

KEY CONCEPTS

WORKING WITH DATABASES

GETTING DATA FROM USERS

DISPLAYING DATA

SECURITY

CACHING

RESTFUL WEB SERVICES

DEVELOPMENT TOOLS

TESTING

SPECIAL TOPICS

Advanced Project Template (https://www.yiiframework.com/extension/yiisoft/yii2-app-advanced/doc/guide)

Building Application from Scratch (/doc/guide/2.0/en/tutorial-start-from-scratch)

Console Commands (/doc/guide/2.0/en/tutorial-console)

Core Validators (/doc/guide/2.0/en/tutorial-core-validators)

Docker (/doc/guide/2.0/en

Mailing

☆ 4 followers

1. Configuration
2. Basic usage
3. Composing mail content
4. File attachment
5. Embedding images
6. Testing and debugging
7. Creating your own mail solution

Note: This section is under development.

Yii supports composition and sending of the email messages. However, the framework core provides only the content composition functionality and basic interface. Actual mail sending mechanism should be provided by the extension, because different projects may require its different implementation and it usually depends on the external services and libraries.

For the most common cases you can use yii2-swiftmailer (https://www.yiiframework.com/extension/yiisoft/yii2-swiftmailer) official extension.

Configuration

Mail component configuration depends on the extension you have chosen. In general your application configuration should look like:

```
return [
    //....
    'components' => [
        'mailer' => [
            'class' => 'yii\swiftmailer\Mailer',
        ],
    ],
];
```

Basic usage

Once the `mailer` component is configured, you can use the following code to send an email message:

```
Yii::$app->mailer->compose()
    ->setFrom('from@domain.com')
    ->setTo('to@domain.com')
    ->setSubject('Message subject')
    ->setTextBody('Plain text content')
    ->setHtmlBody('<b>HTML content</b>')
    ->send();
```

In the above example the method `compose()` creates an instance of the mail message, which then is populated and sent. You may put more complex logic in this process if needed:

```
$message = Yii::$app->mailer->compose();
if (Yii::$app->user->isGuest) {
    $message->setFrom('from@domain.com');
} else {
    $message->setFrom(Yii::$app->user->identity->email);
}
$message->setTo(Yii::$app->params['adminEmail'])
    ->setSubject('Message subject')
    ->setTextBody('Plain text content')
    ->send();
```

/tutorial-docker)

Internationalization (/doc/guide/2.0/en/tutorial-i18n)

Mailing (/doc/guide/2.0/en/tutorial-mailing)

Performance Tuning (/doc/guide/2.0/en/tutorial-performance-tuning)

Shared Hosting Environment (/doc/guide/2.0/en/tutorial-shared-hosting)

Template Engines (/doc/guide/2.0/en/tutorial-template-engines)

Working with Third-Party Code (/doc/guide/2.0/en/tutorial-yii-integration)

Using Yii as a micro framework (/doc/guide/2.0/en/tutorial-yii-as-micro-framework)

WIDGETS

HELPERS

> **Note:** each `mailer` extension comes in 2 major classes: `Mailer` and `Message`. `Mailer` always knows the class name and specific of the `Message`. Do not attempt to instantiate `Message` object directly – always use `compose()` method for it.

You may also send several messages at once:

```
$messages = [];
foreach ($users as $user) {
    $messages[] = Yii::$app->mailer->compose()
        // ...
        ->setTo($user->email);
}
Yii::$app->mailer->sendMultiple($messages);
```

Some particular mail extensions may benefit from this approach, using single network message etc.

Composing mail content

Yii allows composition of the actual mail messages content via special view files. By default these files should be located at `@app/mail` path.

Example mail view file content:

```
<?php
use yii\helpers\Html;
use yii\helpers\Url;

/* @var $this \yii\web\View view component instance */
/* @var $message \yii\mail\BaseMessage instance of newly cr

?>
<h2>This message allows you to visit our site home page by
<?= Html::a('Go to home page', Url::home('http')) ?>
```

In order to compose message content via view file simply pass view name to the `compose()` method:

```
Yii::$app->mailer->compose('home-link') // a view rendering
    ->setFrom('from@domain.com')
    ->setTo('to@domain.com')
    ->setSubject('Message subject')
    ->send();
```

You may pass additional view parameters to `compose()` method, which will be available inside the view files:

```
Yii::$app->mailer->compose('greetings', [
    'user' => Yii::$app->user->identity,
    'advertisement' => $adContent,
]);
```

You can specify different view files for HTML and plain text message contents:

```
Yii::$app->mailer->compose([
    'html' => 'contact-html',
    'text' => 'contact-text',
]);
```

If you specify view name as a scalar string, its rendering result will be used as HTML body, while plain text body will be composed by removing all HTML entities from HTML one.

View rendering result can be wrapped into the layout, which can be setup using yii\mail\BaseMailer::$htmlLayout (/doc/api/2.0/yii-mail-basemailer#$htmlLayout-detail) and yii\mail\BaseMailer::$textLayout (/doc/api/2.0/yii-mail-basemailer#$textLayout-detail). It will work the same way like layouts in regular web application. Layout can be used to setup mail CSS styles or other shared content:

```
<?php
use yii\helpers\Html;

/* @var $this \yii\web\View view component instance */
/* @var $message \yii\mail\MessageInterface the message bei
/* @var $content string main view render result */
?>
<?php $this->beginPage() ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "h
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta http-equiv="Content-Type" content="text/html; cha
    <style type="text/css">
        .heading {...}
        .list {...}
        .footer {...}
    </style>
    <?php $this->head() ?>
</head>
<body>
    <?php $this->beginBody() ?>
    <?= $content ?>
    <div class="footer">With kind regards, <?= Yii::$app->n
    <?php $this->endBody() ?>
</body>
</html>
<?php $this->endPage() ?>
```

File attachment

You can add attachments to message using methods `attach()` and `attachContent()`:

```
$message = Yii::$app->mailer->compose();

// attach file from local file system
$message->attach('/path/to/source/file.pdf');

// create attachment on-the-fly
$message->attachContent('Attachment content', ['fileName' =
```

Embedding images

You can embed images into the message content using `embed()` method. This method returns the attachment id, which should be then used at `img` tag. This method is easy to use when composing message content via view file:

```
Yii::$app->mailer->compose('embed-email', ['imageFileName'
    // ...
    ->send();
```

Then inside the view file you can use the following code:

```
<img src="<?= $message->embed($imageFileName); ?>">
```

Testing and debugging

A developer often has to check, what actual emails are sent by the application, what was their content and so on. Such ability is granted by Yii via `yii\mail\BaseMailer::useFileTransport`. If enabled, this option enforces saving mail message data into the local files instead of regular sending. These files will be saved under `yii\mail\BaseMailer::fileTransportPath`, which is `@runtime/mail` by default.

> **Note:** you can either save the messages to the files or send them to the actual recipients, but can not do both simultaneously.

A mail message file can be opened by a regular text file editor, so you can browse the actual message headers, content and so on. This mechanism may prove itself, while debugging application or running unit test.

> **Note:** the mail message file content is composed via `\yii\mail\MessageInterface::toString()`, so it depends on the actual mail extension you are using in your application.

Creating your own mail solution

In order to create your own custom mail solution, you need to create 2 classes: one for the `Mailer` and another one for the `Message`. You can use `yii\mail\BaseMailer` and `yii\mail\BaseMessage` as the base classes for your solution. These classes already contain the basic logic, which is described in this guide. However, their usage is not mandatory, it is enough to implement `yii\mail\MailerInterface` and `yii\mail\MessageInterface` interfaces. Then you need to implement all the abstract methods to build your solution.

Internationalization (/doc/guide/2.0/en/tutorial-i18n) Go to Top

Performance Tuning (/doc/guide/2.0/en/tutorial-performance-tuning)

User Contributed Notes

Leave a comment

Signup (/signup) or Login (/login) in order to comment.

(http://www.eshill.ru/)

Supported by

Your donations (https://opencollective.com/yiisoft)

(https://www.jetbrains.com/?from=yii)