'class' => 'yii\log\EmailTarget',

'categories' => ['example'],

[

'levels' => ['error'],

// 'mailer' => 'mailer',

'message' => [

'from' => ['log@example.com'],

'to' => ['developer1@example.com', 'developer2@example.com'],

'subject' => 'Log error,

],

],

EmailTarget sends log messages through an e-mail via the Yii: :$app->mailer component by default. We limit category to example and level to error. An e-mail will be sent from log@example. com to two developers and the subject will be Log error:

[

'class' => 'yii\log\FileTarget',

'levels' => [warning],

'logFile' => '@runtime/logs/warning.log',

],

FileTarget appends error messages to a specified file. We limit the message level to warning and use a file named warning. log. We do the same for info-level messages by using a file named Info. log.

Also, we can use yii\log\SyslogTarget to write messages into the Unix /var/log/syslog system file and yii\log\DbTarget to write logs into the database. For the second one, you must apply their migrations:

./yii migrate --migrationPath=@yii/log/migrations/

There’s more...

There are more interesting things about Yii logging, which are covered in the following subsections.

Yii::trace versus Yii::getLogger()->log

Yii: :trace is a simple wrapper around Yii: :log:

public static function trace($message, $category = 'application')

{

if (YII\_DEBUG) {

static::getLogger()->log($message, Logger::LEVEL\_TRACE, $category);

}

}

Therefore, Yii: :trace logs a message with a trace level, if Yii is in the debug mode.

Yii::beginProfile and Yii::endProfile

These methods are used to measure the execution time of some part of the application’s code. In our LogController, we measured 10,000 executions of preg\_replace as follows:

Yii::beginProfile('preg\_replace', ' example');

for($i=0;$i<10000;$i++){