notificationsdelivery Follow @yiiframework

It is easy to deliver notifications to users into different channels (email, sms, etc)

9 followers 0 0 report it

This is powerful modules pack used in high-load applications with many types of notifications, many types of notification delivery channels (email, sms, etc...). It supports "just now" or "background" deliveries without changing of a code.

This extension consists of two modules: 1. Notifications 2. Delivery

Notification is an entity with following general properties: ID, Type, Date Created. There is the abstract notification model class in the "notifications" module - ActiveNotification. There is only one real (not abstract) notification class inside "notification" module: "NotificationStatic". There are two additional properties in the NotificationStatic class: "subject" and "content".

Delivery is an entity with following properties: ID, Notification ID, Recipients. So, delivery contains information about recipients which should receive an assigned notification. Single delivery has one or multiple recipients. Recipient is an entity with following properties: Delivery ID (mandatory), Recipient Credentials (mandatory), Recipient ID (optional), Channel (mandatory), Send Time (optional).

Requirements

1. e.g. Yii 1.1.13 or above.
2. mailer extension with my patch (mailer extension origin http://www.yiiframework.com/extension/mailer) installed. Patched mailer extension is attached into downloads section of this extension. 'mailer' component must be defined in an application config.
3. Models extension http://www.yiiframework.com/extension/models version 1.1 or above.
4. CPHPViewRenderer or other renderer component must be defined in an application config.

Usage

1. Attach custom view renderer component into an application.

2. Copy the mailer extension into "application.extensions" directory. Attach mailer component into an application.

```
...
'import' => array(
    ...
    'application.extensions.mailer.*',
    ...
),
...
```

Create extension

SEARCH EXTENSIONS

Find

Downloads

» delivery.zip
» notifications.zip
» mailer-patched.zip

See all »

Yii Version:
License: New BSD License
Developed by: djvibegga
Category: Others
Votes: *No votes yet*
Downloaded: 614 times
Created on: Nov 18, 2013
Last updated: Nov 18, 2013
Tags: console command, email, sms, cronjobs, module

Related Extensions

extended-database-migration
simplepaypal
signals-module
smigrationcommand
dash

```
'components' => array(
    ...
    'mailer' => array(
        'class' => 'EMailer',
        'Host' => 'your.smtp.host',
        'Port' => 465,
        'SMTPAuth' => true,
        'SMTPSecure' => 'ssl',
        'Username' => 'mail account name',
        'Password' => 'mail account password',
        'SMTPAuth' => true,
        'ContentType' => 'text/html',
        'From'     => 'webmaster.myhost.com',
        'FromName' => 'MyHost Webmaster',
        'CharSet'  => 'UTF-8',
    ),
    ...
),
...
```

Usage of SSL isn't required. But above in example SSL is used.

3. Define models extension import pathes:

```
...
'import' => array(
    ...
    'application.extensions.models.*',
    'application.extensions.models.interfaces.*',
    ...
),
...
```

4. Copy migration directories' content of "notifications" and "delivery" modules into "application.migrations" directory. Apply these migrations.

5. Copy the notification module inside the "application.modules.notifications" directory. Attach notifications module into an application.

```
...
'import' => array(
    ...
    'application.modules.notifications.*',
    'application.modules.notifications.interfaces.*',
    'application.modules.notifications.models.*',
    ...
),
...
'modules' => array(
    ...
    'notifications' => array(
        'class' => 'NotificationsModule',
    ),
    ...
),
...
```

6. Copy the notification module inside the "application.modules.delivery" directory. Attach delivery module into an application.

```
...
```

```
'import' => array(
    ...
    'application.modules.delivery.*',
    'application.modules.delivery.components.*',
    'application.modules.delivery.interfaces.*',
    'application.modules.delivery.models.*',
    'application.modules.delivery.commands.*',
    ...
),
...
'modules' => array(
    ...
    'delivery' => array(
        'class' => 'DeliveryModule',
        'components' => array(
            'manager' => array(
                'class' => 'DeliveryManager',
                'config' => array(
                    'channels' => array(
                        array(
                            'class' => 'MyEmailDeliveryChannel',
                        ),
                    ),
                    'fetch' => array(
                        /*
                        ...
                        NotificationType::NOTIFICATION_REVIEW_CREATED => array(
                             'class' => 'NotificationReviewCreated',
                        ),
                        ...
                        */
                    ),
                ),
            ),
        ),
    ),
    ...
),
...
```

NotificationType::NOTIFICATION_REVIEW_CREATED - is a custom defined notification type (must have integer data type) in your application. 'NotificationReviewCreated' - name of the AR model class used for fetching notifications of type NotificationType::NOTIFICATION_REVIEW_CREATED.

7. Create application-based delivery channels implementations. For example, email delivery channel should looks like below:

```
...
class MyEmailDeliveryChannel extends EmailDeliveryChannel
{
    /**
     * Applies criteria for retrieving email credentials. You must
     * join all tables necessary for fetching email credentials
     * of users. This method would be invoked when background
     * delivery process will fetch delivery items with their
     * recipients.
     *
     * @param CDbCriteria $criteria existing criteria
     *
     * @return void
     * @see FilterModel::apply()
     */
    public function apply($criteria)
    {
        parent::apply($criteria);
        $with = array(
            'recipients.user' => array(
                'alias' => 'recipient_profile',
                'select' => array('recipient_profile.email'),
            ),
            'sender' => array(
                'alias' => 'sender_profile',
                'select' => array('sender_profile.email'),
            ),
        );
        $criteria->with = CMap::mergeArray($criteria->with, $with);
        $criteria->together = true;
    }

    /**
     * Return array of user Credentials. For understanding
     * this method please see examples of usage above. So,
     * you will understand what is the "User" entity.
     *
     * @param User $user recipient user model
     *
     * @return array of user credentials
     * @throws Exception
     */
    public function getUserCredentials($user)
    {
        return array('email' => $user->email);
    }

    /**
     * (non-PHPdoc)
     *
     * @param mixed &$delivery delivery item
     *
     * @return string sender email address
     * @see IDeliveryChannel::getSender($delivery)
     */
    public function getSender(&$delivery)
    {
        return $delivery->from_credentials['email'];
    }

    /**
     * (non-PHPdoc)
     *
     * @param mixed &$delivery delivery item
     *
     * @return array recipient email addresses
     * @see IDeliveryChannel::getRecipients($delivery)
     */
    public function getRecipients(&$delivery)
    {
        $recipients = $delivery->getRecipients($this->getType());
        $ret = array();
        foreach ($recipients as $recipient) {
            $ret[] = $recipient->credentials['email'];
        }
        return $ret;
    }
}
...
```

If you want to do SMS or any other delivery channel you must inherit your class from BaseDeliveryChannel and implement processSend() abstract method.

Oh… I know this is hard now… Be patient, because the delivery process will be very easy and scalable!

Lets try to deliver some static emails to three website users just now:

```
...
$context = $renderer = Yii::app()->getComponent('viewRenderer');
$viewData = array('param1' => 'value1', 'param2' => 'value2');

$notification = new NotificationStatic();
$notification->subject = 'Hello. This is test subject.';
$notification->content = $renderer->renderFile($context, 'absolute/path/to/email
/template', $viewData, true);

Yii::app()->getModule('notifications')
    ->getComponent('manager')
    ->add($notification);//if you don't want to track this notification in the database
then remove these lines.

$criteria = new CDbCriteria();
$criteria->limit = 3;
$recipients = User::model()->findAll($criteria);
$sender = null;//if you pass undefined sender credentials, then system applies global
sender credentials defined for selected delivery channel

if (!DeliverableNotification::quickDeliver(
    $notification,
    $sender, $recipients, IDeliveryChannel::TYPE_EMAIL,
    DeliveryPriority::JUST_NOW
) {
    $message = Yii::t('core', 'Can\'t send notification. Notification type: "' .
get_class($notification) . '".');
    Yii::log($message, CLogger::LEVEL_ERROR);
}
...
```

So, code example above would deliver notifications synchronously (script will wait until all notification was sent). If time of a website page response is critical you can add delivery item into delivery queue with "high", "medium" or "low" priority. So, in this way notification has been delivered in background (when cron job has been invoked).

```
...
//notification prepare same as above
...
if (!DeliverableNotification::quickDeliver(
    $notification,
    $sender, $recipients, IDeliveryChannel::TYPE_EMAIL,
    DeliveryPriority::HIGH
) {
    $message = Yii::t('core', 'Can\'t push notification into delivery queue. Notification
type: "' . get_class($notification) . '".');
    Yii::log($message, CLogger::LEVEL_ERROR);
}
...
```

Here is console command template used for processing background delivery:

```
yiic delivery dequeue [--priority=<priority>] [--limit=<limit>] [--channel=<channel>]
```

To do background delivery you need to add all above configurations into the console.php and run add into crontab following:

```
*/5 * * * * /path/to/yiic delivery dequeue low 100 email
*/2 * * * * /path/to/yiic delivery dequeue medium 80 email
* * * * * /path/to/yiic delivery dequeue high 60 email
```

So, these jobs above simulates delivery sending priorities. If you have an SMS channel implementation. You can add into cron something like that:

```
*/3 * * * * /path/to/yiic delivery dequeue high 200 sms
```

Also, you can create custom notifications. It is better to create custom notification class instead of writing copy paste… See example below:

```
class NotificationLowCreditBallance extends DeliverableNotification
{
    /**
     * Returns the static model of the specified AR class.
     *
     * @param string $className the name of the model class
     *
     * @return NotificationLowCreditBallance the static model class
     */
    public static function model($className = __CLASS__)
    {
        return parent::model($className);
    }

    /**
     * (non-PHPdoc)
     *
     * @return string table name
     * @see NotificationStatic::tableName()
     */
    public function tableName()
    {
        return 'notification_low_credit_ballance';
    }

    /**
     * Custom validation rules
     *
     * @return void
     */
    public function rules()
    {
        return array_merge(
             parent::rules(),
             array(
                 ..
                 array('balance, user_id', 'required'),
                 array('balance', 'numerical', 'min' => 0),
                 …
             )
        );
    }

    /**
     * (non-PHPdoc)
     *
     * @return relations
     * @see ActiveNotification::relations()
     */
```

```
        public function relations()
        {
            return array_merge(
                parent::relations(),
                array(
                    'user' => array(self::BELONGS_TO, 'User', 'user_id'),
                )
            );
        }

        /**
         * Returns notification type
         *
         * @return integer notification type
         */
        public function getType()
        {
            return NotificationType::TYPE_LOW_CREDIT_BALLANCE;
        }

        /**
         * Handles delivering notification. You can switch subject and content will be
         * delivered depending on active channel.
         *
         * @param IDeliveryChannel $channel   channel instance which sends delivery data
         * @param string           $sender    notification sender
         * @param string           $recipient notification recipient
         *
         * @return boolean whether notificaation delivered successfully
         * @see NotificationStatic::deliver()
         */
        public function deliver(IDeliveryChannel $channel, $sender, $recipient)
        {
            $subject = Yii::app()->name . ' - '
                     . Yii::t('NotificationLowCreditBalance', 'Your credits are getting low.');
            $viewData = array('user' => $this->user);
            $content = DeliverableNotification::renderContent(
                'application.views.emails.low_credit_balance', $viewData
            );
            return $channel->send($sender, $recipient, $subject, $content);
        }
    }
```

Resources

- http://blog.jviba.com

Total 2 comments

#15529 djvibegga at 2013/11/19 04:17am 0 0 report it

Re: open source

I haven't those modules on the Github. But it is a good idea to create a fork there. I can do that ASAP. Thanks!

#15525 nineinchnick at 2013/11/19 01:16am 0 0 report it

open source

I guess this extension is open source? Do you have a public repository like on Github?

I wrote my own notifications extensions and maybe we would compare ideas. I made mine mainly for web notifications (html5).

Leave a comment

Please login to leave your comment.

Yii Supporters

The Ext4Yii Framework is a professional PHP Yii extension which provides server-side ExtJS functionality.

Copyright © 2014 by Yii Software LLC. All Rights Reserved.