django-comments-xtd Documentation

Release 1.3a1

Daniel Rus Morales

March 04, 2014

Contents

django-comments-xtd extends the built-in Django's Comments Framework with:

1. Thread support, so comments may be nested
2. The maximum thread level can be set up either for all models or on a per app.model basis
3. Optional notification of follow-up comments via email
4. Mute links on follow-up emails to allow follow-up notification cancellation
5. Comment confirmation via email when users are not authenticated
6. Comments hit the database only when have been confirmed
7. Template tags to list/render the last N comments posted to any given list of app.model pairs
8. Comments can be formatted in Markdown, reStructuredText, linebreaks or plain text
9. Emails sent through threads (can be disable to allow other solutions, like a Celery app)

CHAPTER 1

Demo projects

Django-comments-xtd comes with three demo projects:

1. **simple**: Single model with **non-threaded** comments
2. **simple_threads**: Single model with **threaded** comments up to level 2
3. **multiple**: Several models with comments, and a maximum thread level defined for each app.model pair.

Click here for a quick look at the examples directory in the repository.

1.1 Demo sites setup

The recommended way to run the demo sites is in its own virtualenv. Once in a new virtualenv, clone the code and cd into any of the 3 demo sites. Then run the install script and launch the dev server:

```
$ git clone git://github.com/danirus/django-comments-xtd.git
$ cd django-comments-xtd/django_comments_xtd/demos/[simple|simple_thread|multiple]
$ sh ./install.sh (to syncdb, migrate and loaddata)
$ python manage.py runserver
```

By default:

- There's an `admin` user, with password `admin`
- Emails are sent to the `console.EmailBackend`. Comment out `EMAIL_BACKEND` in the settings module to send actual emails.

1.2 Simple demo site

The **simple** demo site is a project with just one application called **articles** with an **Article** model whose instances accept comments. The example features:

- Comments have to be confirmed by email before they hit the database.
- Users may request follow-up notifications.
- Users may cancel follow-up notifications by clicking on the mute link.

Follow the next steps to give them a try:

1. Visit http://localhost:8000/ and look at your articles' detail page.

2. Log out of the admin site to post comments, otherwise they will be automatically confirmed and no email will be sent.
3. When adding new articles in the admin interface be sure to tick the box *allow comments*, otherwise comments won't be allowed.
4. Send new comments with the Follow-up box ticked and a different email address. You won't receive follow-up notifications for comments posted from the same email address the new comment is being confirmed from.
5. Click on the Mute link on the Follow-up notification email and send another comment.

1.3 Simple with threads

The **simple_threads** demo site extends the **simple** demo functionality featuring:

- Thread support up to level 2

1. Visit http://localhost:8000/ and look at the first article page with 9 comments.
2. See the comments in the admin interface too:

- The first field represents the thread level.
- When in a nested comment the first field refers to the parent comment.

1.4 Multiple demo site

The **multiple** demo allows users post comments to three different type of instances: stories, quotes, and releases. Stories and quotes belong to the **blog app** while releases belong to the **projects app**. The demo shows the blog homepage with the last 5 comments posted to either stories or quotes and a link to the complete paginated list of comments posted to the blog. It features:

- Definition of maximum thread level on a per app.model basis.
- Use of comments_xtd template tags, `get_xtdcomment_count`, `render_last_xtdcomments`, `get_last_xtdcomments`, and the filter `render_markup_comment`.

1. Visit http://localhost:8000/ and take a look at the **Blog** and **Projects** pages.

- The **Blog** contains **Stories** and **Quotes**. Instances of both models have comments. The blog index page shows the **last 5 comments** posted to either stories or quotes. It also gives access to the **complete paginated list of comments**.
- Project releases have comments as well but are not included in the complete paginated list of comments shown in the blog.

2. To render the last 5 comments the site uses:
   - The templatetag `{% render_last_xtdcomments 5 for blog.story blog.quote %}`
   - And the following template files from the `demos/multiple/templates` directory:
   - `django_comments_xtd/blog/story/comment.html` to render comments posted to **stories**
   - `django_comments_xtd/blog/quote/comment.html` to render comments posted to **quotes**
   - You may rather use a common template to render comments:

   - For all blog app models: `django_comments_xtd/blog/comment.html`
   - For all the website models: `django_comments_xtd/comment.html`

3. To render the complete paginated list of comments the site uses:

   - An instance of a generic `ListView` class declared in `blog/urls.py` that uses the following queryset:
   - `XtdComment.objects.for_app_models("blog.story", "blog.quote")`

4. The comment posted to the story **Net Neutrality in Jeopardy** starts with a specific line to get the content rendered as reStructuredText. Go to the admin site and see the source of the comment; it's the one sent by Alice to the story 2.

   - To format and render a comment in a markup language, make sure the first line of the comment looks like: `#!<markup-language>` being `<markup-language>` any of the following options:
   - markdown
   - restructuredtext
   - linebreaks
   - Then use the filter `render_markup_comment` with the comment field in your template to interpret the content (see `demos/multiple/templates/comments/list.html`).

CHAPTER 2

Tutorial

Django-comments-xtd is a reusable app that extends the built-in Django Comments Framework.

2.1 Installation

Check out the code and add it to your project or `PYTHONPATH`. Use git, pip or easy_install to check out Django-comments-xtd from Github or get a release from PyPI:

1. Use **git** to clone the repository, and then install the package (read more about git):

- `git clone git://github.com/danirus/django-comments-xtd.git` and
- `python setup.py install`

2. Or use **pip** (read more about pip):

- Do `pip install django-comments-xtd`, or
- Edit your project's `requirements` file and append either the Github URL or the package name `django-comments-xtd`, and then do `pip install -r requirements`.

3. Or use **easy_install** (read more about easy_install):

- Do `easy_install django-comments-xtd`

4. Optionally, if you want to allow comments written in markup languages like Markdown or reStructuredText, install django-markup.

2.2 Configuration

Configuring Django-comments-xtd comprehends the following steps:

1. Add `COMMENTS_APP = "django_comments_xtd"` to your settings module.
2. Add `'django.contrib.comments'` and `'django_comments_xtd'` to your `INSTALLED_APPS` setting.

- If you allow comments written in markup languages add `django_markup` to `INSTALLED_APPS` too.

3. Add `COMMENTS_XTD_CONFIRM_EMAIL = True` to the settings file in order to require comment confirmation by email for not logged-in users.
4. Add `url(r'^comments/', include('django_comments_xtd.urls'))` to your `urls.py`.

5. Create a `comments` directory in your templates directory and copy those templates from the Django Comments Framework that you want to customise. Following are the most important:

- `comments/list.html` used by templatetag `render_comments_list`
- `comments/form.html` used by templatetag `render_comment_form`
- `comments/preview.html` used to preview the comment or when there are form errors
- `comments/posted.html` rendered after comment is sent

6. To customise how templatetag `render_last_xtdcomments` renders comments, copy the template file `django_comment_xtd/comment.html` to any of the following targets in your `templates` directory:

- `django_comment_xtd/<app>/<model>/comment.html` to customise them for the given `<app>.<model>`
- `django_comment_xtd/<app>/comment.html` to customise them for all `<app>` models
- `django_comment_xtd/comment.html` to customise all your site comments at once

7. Run `python manage.py syncdb` to create the `django_comments_xtd_xtdcomment` table.

Optionally you can add extra settings to control Django-comments-xtd behaviour (see *Settings*). They all have sane values by default.

2.3 Workflow

Here is the application workflow described in 4 actions:

1. The user visits a page that accepts comments. Your app or a 3rd. party app handles the request:

1. Your template shows content that accepts comments. It loads the `comments` templatetag and using tags as `render_comment_list` and `render_comment_form` the template shows the current list of comments and the *post your comment* form.

2. The user **clicks on preview**. Django Comments Framework `post_comment` view handles the request:

1. Renders `comments/preview.html` either with the comment preview or with form errors if any.

3. The user **clicks on post**. Django Comments Framework `post_comment` view handles the request:

    1. If there were form errors it does the same as in point 2.
    2. Otherwise creates an instance of `TmpXtdComment` model: an in-memory representation of the comment.
    3. Send signal `comment_will_be_posted` and `comment_was_posted`. The *django-comments-xtd* receiver `on_comment_was_posted` receives the second signal with the `TmpXtdComment` instance and does as follows:

    - If the user is authenticated or confirmation by email is not required (see *Settings*):
        - An instance of `XtdComment` hits the database.
        - An email notification is sent to previous comments followers telling them about the new comment following up theirs. Comment followers are those who ticked the box *Notify me of follow up comments via email*.
    - Otherwise a confirmation email is sent to the user with a link to confirm the comment. The link contains a secured token with the `TmpXtdComment`. See below *Creating the secure token for the confirmation URL*.

4. Pass control to the `next` parameter handler if any, or render the `comments/posted.html` template:
   - If the instance of `XtdComment` has already been created, redirect to the the comments's absolute URL.
   - Otherwise the template content should inform the user about the confirmation request sent by email.

4. The user **clicks on the confirmation link**, in the email message. *Django-comments-xtd* `confirm` view handles the request:

1. Checks the secured token in the URL. If it's wrong returns a 404 code.
2. Otherwise checks whether the comment was already confirmed, in such a case returns a 404 code.
3. Otherwise sends a `confirmation_received` signal. You can register a receiver to this signal to do some extra process before approving the comment. See *Signal and receiver*. If any receiver returns False the comment will be rejected and the template `django_comments_xtd/discarded.html` will be rendered.
4. Otherwise an instance of `XtdComment` finally hits the database, and
5. An email notification is sent to previous comments followers telling them about the new comment following up theirs.

2.3.1 Creating the secure token for the confirmation URL

The Confirmation URL sent by email to the user has a secured token with the comment. To create the token Django-comments-xtd uses the module `signed.py` authored by Simon Willison and provided in Django-OpenID.

`django_openid.signed` offers two high level functions:

- **dumps**: Returns URL-safe, sha1 signed base64 compressed pickle of a given object.
- **loads**: Reverse of dumps(), raises ValueError if signature fails.

A brief example:

```
>>> signed.dumps("hello")
'UydoZWxsbycKcDAKLg.QLtjWHYe7udYuZeQyLlafPqAx1E'

>>> signed.loads('UydoZWxsbycKcDAKLg.QLtjWHYe7udYuZeQyLlafPqAx1E')
'hello'

>>> signed.loads('UydoZWxsbycKcDAKLg.QLtjWHYe7udYuZeQyLlafPqAx1E-modified')
BadSignature: Signature failed: QLtjWHYe7udYuZeQyLlafPqAx1E-modified
```

There are two components in dump's output `UydoZWxsbycKcDAKLg.QLtjWHYe7udYuZeQyLlafPqAx1E`, separatad by a '.'. The first component is a URLsafe base64 encoded pickle of the object passed to dumps(). The second component is a base64 encoded hmac/SHA1 hash of "$first_component.$secret".

Calling signed.loads(s) checks the signature BEFORE unpickling the object -this protects against malformed pickle attacks. If the signature fails, a ValueError subclass is raised (actually a BadSignature).

2.4 Signal and receiver

In addition to the signals sent by the Django Comments Framework, django-comments-xtd sends the following signal:

- **confirmation_received**: Sent when the user clicks on the confirmation link and before the `XtdComment` instance is created in the database.

- **comment_thread_muted**: Sent when the user clicks on the mute link, in a follow-up notification.

2.4.1 Sample use of the `confirmation_received` signal

You might want to register a receiver for `confirmation_received`. An example function receiver could check the time stamp in which a user submitted a comment and the time stamp in which the confirmation URL has been clicked. If the difference between them is over 7 days we will discard the message with a graceful *"sorry, it's a too old comment"* template.

Extending the demo site with the following code will do the job:

```
#----------------------------------------
# append the code below to demo/views.py:

from datetime import datetime, timedelta
from django_comments_xtd import signals

def check_submit_date_is_within_last_7days(sender, data, request, **kwargs):
    plus7days = timedelta(days=7)
    if data["submit_date"] + plus7days < datetime.now():
        return False
signals.confirmation_received.connect(check_submit_date_is_within_last_7days)


#-----------------------------------------------------
# change get_comment_create_data in django_comments_xtd/forms.py to cheat a
# bit and make Django believe that the comment was submitted 7 days ago:

def get_comment_create_data(self):
    from datetime import timedelta                                     # ADD THIS

    data = super(CommentForm, self).get_comment_create_data()
    data['followup'] = self.cleaned_data['followup']
    if settings.COMMENTS_XTD_CONFIRM_EMAIL:
        # comment must be verified before getting approved
        data['is_public'] = False
    data['submit_date'] = datetime.datetime.now() - timedelta(days=8)   # ADD THIS
    return data
```

Try the demo site again and see that the *django_comments_xtd/discarded.html* template is rendered after clicking on the confirmation URL.

2.5 Maximum Thread Level

Nested comments are disabled by default, to enable them use the following settings:

- `COMMENTS_XTD_MAX_THREAD_LEVEL`: an integer value
- `COMMENTS_XTD_MAX_THREAD_LEVEL_BY_APP_MODEL`: a dictionary

Django-comments-xtd inherits the flexibility of the built-in Django Comments Framework, so that developers can plug it to support comments on as many models as they want in their projects. It is as suitable for one model based project, like comments posted to stories in a simple blog, as for a project with multiple applications and models.

The configuration of the maximum thread level on a simple project is done by declaring the `COMMENTS_XTD_MAX_THREAD_LEVEL` in the `settings.py` file:

```
COMMENTS_XTD_MAX_THREAD_LEVEL = 2
```

Comments then could be nested up to level 2:

```
<In an instance detail page that allows comments>

First comment (level 0)
  |-- Comment to First comment (level 1)
    |-- Comment to Comment to First comment (level 2)
```

Comments posted to instances of every model in the project will allow up to level 2 of threading.

On a project that allows users posting comments to instances of different models, the developer may want to declare a maximum thread level per `app.model` basis. For example, on an imaginary blog project with stories, quotes, diary entries and book/movie reviews, the developer might want to define a default project wide maximum thread level of 1 for any model and an specific maximum level of 5 for stories and quotes:

```
COMMENTS_XTD_MAX_THREAD_LEVEL = 1
COMMENTS_XTD_MAX_THREAD_LEVEL_BY_APP_MODEL = {
    'blog.story': 5,
    'blog.quote': 5,
}
```

So that `blog.review` and `blog.diaryentry` instances would support comments nested up to level 1, while `blog.story` and `blog.quote` instances would allow comments nested up to level 5.

CHAPTER 3

Filters and Template Tags

Django-comments-xtd comes with three tags and one filter:

- Tag `get_xtdcomment_count`
- Tag `get_last_xtdcomments`
- Tag `render_last_xtdcomments`
- Filter `render_markup_comment`

To use any of them in your templates you first need to load them:

```
{% load comments_xtd %}
```

3.1 Get Xtdcomment Count

Tag syntax:

```
{% get_xtdcomment_count as [varname] for [app].[model] [[app].[model] ...] %}
```

Gets the comment count for the given pairs `<app>.<model>` and populates the template context with a variable containing that value, whose name is defined by the `as` clause.

3.1.1 Example usage

Get the count of comments the model `Story` of the app `blog` have received, and store it in the context variable `comment_count`:

```
{% get_xtdcomment_count as comment_count for blog.story %}
```

Get the count of comments two models, `Story` and `Quote`, have received and store it in the context variable `comment_count`:

```
{% get_xtdcomment_count as comment_count for blog.story blog.quote %}
```

3.2 Get Last Xtdcomments

Tag syntax:

```
{% get_last_xtdcomments [N] as [varname] for [app].[model] [[app].[model] ...] %}
```

Gets the list of the last N comments for the given pairs `<app>.<model>` and stores it in the template context whose name is defined by the `as` clause.

3.2.1 Example usage

Get the list of the last 10 comments two models, `Story` and `Quote`, have received and store them in the context variable `last_10_comment`. You can then loop over the list with a `for` tag:

```
{% get_last_xtdcomments 10 as last_10_comments for blog.story blog.quote %}
{% if last_10_comments %}
  {% for comment in last_10_comments %}
    <p>{{ comment.comment|linebreaks }}</p> ...
  {% endfor %}
{% else %}
  <p>No comments</p>
{% endif %}
```

3.3 Render Last Xtdcomments

Tag syntax:

```
{% render_last_xtdcomments [N] for [app].[model] [[app].[model] ...] %}
```

Renders the list of the last N comments for the given pairs `<app>.<model>` using the following search list for templates:

- `django_comments_xtd/<app>/<model>/comment.html`
- `django_comments_xtd/<app>/comment.html`
- `django_comments_xtd/comment.html`

3.3.1 Example usage

Render the list of the last 5 comments posted, either to the blog.story model or to the blog.quote model. See it in action in the *Multiple Demo Site*, in the *blog homepage*, template `blog/homepage.html`:

```
{% render_last_xtdcomments 5 for blog.story blog.quote %}
```

3.4 Render Markup Comment

Filter syntax:

```
{{ comment.comment|render_markup_comment }}
```

Renders a comment using a markup language specified in the first line of the comment.

3.4.1 Example usage

A comment like:

```
comment = r'''#!markdown\n\rAn [example](http://url.com/ "Title")'''
```

Would be rendered as a markdown text, producing the output:

```
<p><a href="http://url.com/" title="Title">example</a></p>
```

Markup languages available are:

- Markdown (use `#!markdown`)
- reStructuredText (use `#!restructuredtext`)
- Linebreaks (use `#!linebreaks`)

CHAPTER 4

Settings

In order to use Django-comments-xtd it is required to declare the setting COMMENTS_APP:

```
COMMENTS_APP = "django_comments_xtd"
```

A number of additional settings are available to customize django-comments-xtd behaviour.

4.1 Maximum Thread Level

COMMENTS_XTD_MAX_THREAD_LEVEL - Maximum Thread Level

Optional

Indicate the maximum thread level for comments.

An example:

```
COMMENTS_XTD_MAX_THREAD_LEVEL = 8
```

Defaults to 0. What means threads are not permitted.

4.2 Maximum Thread Level per App.Model

COMMENTS_XTD_MAX_THREAD_LEVEL_BY_APP_MODEL - Maximum Thread Level per app.model basis

Optional

A dictionary with *app_label.model* as keys and the maximum thread level for comments posted to instances of those models as values. It allows definition of max comment thread level on a per *app_label.model* basis.

An example:

```
COMMENTS_XTD_MAX_THREAD_LEVEL = 0
COMMENTS_XTD_MAX_THREAD_LEVEL_BY_MODEL = {
    'projects.release': 2,
    'blog.stories': 8, 'blog.quotes': 8,
    'blog.diarydetail': 0 # not required as it defaults to COMMENTS_XTD_MAX_THREAD_LEVEL
}
```

4.3 Confirm Comment Post by Email

COMMENTS_XTD_CONFIRM_EMAIL - Confirm Comment Post by Email

Optional

This setting establishes whether a comment confirmation should be sent by email. If set to True a confirmation message is sent to the user with a link she has to click on. If the user is already authenticated the confirmation is not sent.

If is set to False the comment is accepted (unless your discard it by returning False when receiving the signal `comment_will_be_posted`, defined by the Django Comments Framework).

An example:

```
COMMENTS_XTD_CONFIRM_EMAIL = True
```

Defaults to True.

4.4 Comment Form Class

COMMENTS_XTD_FORM_CLASS - Form class to use when rendering comment forms.

Optional

A classpath to the form class that will be used for comments.

An example:

```
COMMENTS_XTD_FORM_CLASS = "mycomments.forms.MyCommentForm"
```

Defaults to *"django_comments_xtd.forms.XtdCommentForm"*.

4.5 Comment Model

COMMENTS_XTD_MODEL - Model to use

Optional

A classpath to the model that will be used for comments.

An example:

```
COMMENTS_XTD_MODEL = "mycomments.models.MyCommentModel"
```

Defaults to *"django_comments_xtd.models.XtdComment"*.

4.6 Salt

COMMENTS_XTD_SALT - Extra key to salt the form

Optional

This setting establishes the ASCII string extra_key used by `signed.dumps` to salt the comment form hash. As `signed.dumps` docstring says, just in case you're worried that the NSA might try to brute-force your SHA-1 protected secret.

An example:

```
COMMENTS_XTD_SALT = 'G0h5gt073h6gH4p25GS2g5AQ25hTm256yGt134tMP5TgCX$&HKOYRV'
```

Defaults to an empty string.

4.7 Send HTML Email

COMMENTS_XTD_SEND_HTML_EMAIL - Enable/Disable HTML email messages

Optional

This boolean setting establishes whether email messages have to be sent in HTML format. By the default messages are sent in both Text and HTML format. By disabling the setting email messages will be sent only in Text format.

An example:

```
COMMENTS_XTD_SEND_HTML_EMAIL = True
```

Defaults to True.

4.8 Threaded Emails

COMMENTS_XTD_THREADED_EMAILS - Enable/Disable sending emails in separeate threads

Optional

For low traffic websites sending emails in separate threads is a fine solution. However, for medium to high traffic websites such overhead could be reduce by using other solutions, like a Celery application.

An example:

```
COMMENTS_XTD_THREADED_EMAILS = True
```

Defaults to True.

CHAPTER 5

Templates

List of template files used by django-comments-xtd.

django_comments_xtd/email_confirmation_request.(html|txt) (included) Comment confirmation email sent to the user when she clicks on "Post" to send the comment. The user should click on the link in the message to confirm the comment. If the user is authenticated the message is not sent, as the comment is directly approved.

django_comments_xtd/discarded.html (included) Rendered if any receiver of the signal `confirmation_received` returns False. Tells the user the comment has been discarded.

django_comments_xtd/email_followup_comment.(html|txt) (included) Email message sent when there is a new comment following up the user's. To receive this email the user must tick the box *Notify me of follow up comments via email*.

django_comments_xtd/max_thread_level.html (included) Rendered when a nested comment is sent with a thread level over the maximum thread level allowed. The template in charge of rendering the list of comments can make use of `comment.allow_thread` (True when the comment accepts nested comments) to avoid adding the link "Reply" on comments that don't accept nested comments. See the simple_threads demo site, template `comment/list.html` to see a use example of `comment.allow_thread`.

django_comments_xtd/comment.html (included) Rendered when a logged in user sent a comment via Ajax. The comment gets rendered immediately. JavaScript client side code still has to handle the response.

django_comments_xtd/posted.html Rendered when a not logged-in user sends a comment. Django-comments-xtd try first to find the template in its own template directory, `django_comments_xtd/posted.html`. If it doesn't exist, it will use the template in Django Comments Framework directory: `comments/posted.html`. Look at the demo sites for sample uses.

django_comments_xtd/reply.html (included) Rendered when a user clicks on the *reply* link of a comment. Reply links are created with `XtdComment.get_reply_url` method.

django_comments_xtd/muted.html (included) Rendered when a user clicks on the *mute link* of a follow-up email message.

CHAPTER 6

Quick start

1. In your `settings.py`:
    - Add `django.contrib.comments` and `django_comments_xtd` to `INSTALLED_APPS`
    - Add `COMMENTS_APP = "django_comments_xtd"`
    - Add `COMMENTS_XTD_MAX_THREAD_LEVEL = N`, being `N` the maximum level up to which comments can be threaded:
        - When N = 0: comments are not nested
        - When N = 1: comments can be besited at level 0
        - When N = K: comments can be nested up until level K-1

      This setting can also be set up on a per `<app>.<model>` basis so that you can enable different thread levels for different models. ie: no nested comment for blog posts, up to one thread level for book reviews...

      Read more about `COMMENTS_XTD_MAX_THREAD_LEVEL_BY_APP_MODEL` in the *Tutorial* and see it in action in the **multiple** demo site in *Demo projects*.
    - Customize your project's email settings:
    - `EMAIL_HOST = "smtp.mail.com"`
    - `EMAIL_PORT = "587"`
    - `EMAIL_HOST_USER = "alias@mail.com"`
    - `EMAIL_HOST_PASSWORD = "yourpassword"`
    - `DEFAULT_FROM_EMAIL = "Helpdesk <helpdesk@yourdomain>"`
2. If you want to allow comments written in markup languages like Markdown or reStructuredText:
- Get the dependencies: django-markup
- And add `django_markup` to `INSTALLED_APPS`
3. Add `url(r'^comments/', include('django_comments_xtd.urls'))` to your root URLconf.
4. Change templates to introduce comments:
    - Load the `comments` templatetag and use their tags (ie: in your `templates/app/model_detail.html` template):
    - `{% get_comment_count for object as comment_count %}`
    - `{% render_comment_list for object %}` (uses `comments/list.html`)

- `{% render_comment_form for post %}` (uses `comments/form.html` and `comments/preview.html`)
- Load the `comments_xtd` templatetag and use their tags and filter:
- `{% get_xtdcomment_count as comments_count for blog.story blog.quote %}`
- `{% render_last_xtdcomments 5 for blog.story blog.quote using "blog/comment.html" %}`
- `{% get_last_xtdcomments 5 as last_comments for blog.story blog.quote %}`
- Filter render_markup_comment: `{{ comment.comment|render_markup_comment }}`. You may want to copy and change the template `comments/list.html` from `django.contrib.comments` to use this filter.

5. `syncdb`, `runserver`, and
6. Hit your App's URL!
7. Have questions? Keep reading, and look at the 3 demo sites.

CHAPTER 7

Indices and tables

- *genindex*
- *search*