

О, горячие клавиши заработали!

заккрыть (3)

сегодня в 11:21

Минималистичный issue tracker на Django tutorial

Django*, Python*, Управление проектами*

В этой статье рассказывается, как за короткое время решить с помощью фреймворка Django, простую, но интересную задачу: создание системы баг-трекинга (система тикетов) для своего проекта. Наша система будет интегрирована с системой аутентификации пользователей Django и административным интерфейсом. Несмотря на свою примитивность, такое приложение, при некотором усовершенствовании, вполне может быть полезно для разработчика-одиночки или небольшой группы.

Bugtracker

RegisterLogin

- Menu item
- Menu item
- Menu item

Bug list

Title	Created	Author	Status
Редактирование багрепортов	Sept. 3, 2014	admin	CLOSED
Просмотр багрепортов в виде списка	Sept. 3, 2014	admin	CLOSED
Просмотр конкретного багрепорта	Sept. 3, 2014	admin	CLOSED
Возможность регистрации в качестве нового пользователя	Sept. 3, 2014	admin	CLOSED
Login и Logout	Sept. 3, 2014	admin	CLOSED
Добавление багрепортов	Sept. 3, 2014	admin	CLOSED
Сделать нарядно	Sept. 3, 2014	admin	CLOSED
Создать еще тикетов	Sept. 3, 2014	admin	OPENED

Я опишу действия в linux-окружении, используя Django версии 1.6, поэтому надо иметь ввиду, что для других операционных систем и версий фреймворка, что-то может работать по-другому (но без существенных изменений). Уровень статьи рассчитан на новичков, однако заострять внимание на подготовке рабочего окружения и разжевывать совсем элементарные вещи я не буду, и, если вам не понятно, что вообще делать вначале, рекомендую почитать [вот эту прекрасную статью](#) и пройти [Django tutorial](#).

Итак, какие задачи должно выполнять наше приложение:

- Редактирование багрепортов (доступно для администраторов, через стандартную админку Django)
- Просмотр багрепортов в виде списка и по отдельности (доступно для всех посетителей)
- Возможность регистрации в качестве нового пользователя
- Соответственно, login и logout через веб-интерфейс
- Добавление багрепортов через веб-интерфейс (доступно только для залогиненных пользователей)

В целях соблюдения концепции минималистичности багрепорт (или тикет) будет содержать название (title), описание (description), дату и время создания (created), автора (author), и статус открыт\закрыт (closed).

Процесс работы над приложением будет разделен на несколько этапов, и, чтобы упростить перемещение от одного этапа к другому, я создал [репозиторий](#) на GitHub со всеми исходными кодами и коммитами на каждом этапе. Конечно, можно просто делать копипаст кода из статьи, но я предлагаю поступить иначе:

1. Подготавливаете ваше рабочее окружение, устанавливаете django
2. Если у вас еще нет аккаунта на GitHub — создаете его
3. Делаете [форк \(fork\)](#) моего репозитория
4. В рабочей папке (в шелле) выполняете команду `git clone <url вашего репозитория>`
5. Далее, в начале каждого этапа, я буду указывать команду `git checkout -f <part-#>`, которую необходимо выполнить, чтобы код в рабочей папке синхронизировался с соответствующим коммитом репозитория.

Вы получите полный код готового проекта, в вашей рабочей среде. Можно запустить

```
cd django-tutorial-bugreport
./manage.py runserver
```

и в браузере перейти по адресу [127.0.0.1:8000/bugs](#) «пощупать» финальный результат.

Все möglich в одном месте

QA-сервис для разработчиков

Заказы для фрилансеров

Уютная и дружелюбная

Этап №0 — создание приложения

Прелесть git в том, что можно легко откатиться на любой этап разработки проекта, и для вашего удобства, я эти этапы пометил тегами. Итак, выполняем команду:

```
git checkout -f part-0
```

— тем самым сбросив проект к начальному состоянию, каким бы он был, если бы вы только что выполнили команду `django-admin.py startproject...`

Перейдем в папку проекта и создадим приложение `bugtracker`

```
./manage.py startapp bugtracker
```

Вносим наше приложение в `project/settings.py` (добавив `'bugtracker'` в кортеж `INSTALLED_APPS`)

Сразу настроим каталоги для темплейтов и статики:

```
TEMPLATE_DIRS = (os.path.join(BASE_DIR, 'templates/'),) STATICFILES_DIRS = (os.path.join(BASE_DIR, 'static/'),)
```

Добавим такой `urlpatterns` в `project/urls.py` — `url(r'^bugs/', include('bugtracker.urls'))`, после чего, все остальные нужные нам урлы уже будем описывать именно в `bugtracker\urls.py`, который мы создадим вот таким, пока почти пустым, иначе django будет выдавать ошибку:

```
# coding: utf-8
from django.conf.urls import patterns, include, url

urlpatterns = patterns('',
)
```

Затем создаем модель для багрепорта в файле `bugtracker/models.py`:

```
# coding: utf-8
from django.db import models
from django.contrib.auth.models import User

class Ticket(models.Model):
    title = models.CharField(max_length=128)
    text = models.TextField()
    created = models.DateTimeField(auto_now=True)
    closed = models.BooleanField(default=False)
    user = models.ForeignKey(User,)

    def __unicode__(self):
        return self.title
```

Рассмотрим ее поближе: **title** — это название тикета, например «*Пепелац не взлетает*», **text** — описание и шаги для воспроизведения бага, **created** — время и дата создания тикета, будет автоматически заполняться благодаря `auto_now=True`, **closed** — простейший вариант описания статуса тикета, закрыт он или открыт, по-умолчанию (при создании) `closed = False`, то есть тикет не закрыт, **user** — пользователь, сообщивший о баге. В качестве юникодного представления модель возвращает **title**

Отредактируем `bugtracker/admin.py`, чтобы появилась возможность управлять тикетами из админки:

```
# coding: utf-8
from django.contrib import admin
from models import Ticket

admin.site.register(Ticket)
```

И запустим

```
./manage.py syncdb
```

В результате этого, в базе данных создадутся необходимые таблицы, и будут запрошены логин, e-mail и пароль для создания

пользователя-администратора.

Внимание, далее по тексту, при команде `git checkout` из `git`-репозитория `db.sqlite3`, и администратор имеет логин: `admin`, пароль: `123` (один-два-три)

Создав его, можно запускать:

```
./manage.py runserver
```

И заходить по адресу <http://127.0.0.1:8000/admin>, а после ввода логина и пароля, создавать, редактировать и удалять тикеты.

Этап №1 — список тикетов

Выполняем:

```
git checkout -f part-1
```

— по этой команде, появится файл `db.sqlite3` содержащий ту самую базу данных, в которой уже созданы некоторые тикеты для тестирования, и логин администратора: `admin`, пароль: `123`. Посмотрев содержимое тикетов, вы увидите, что я использовал небогатый функционал нашего приложения в качестве `TODO`-списка для него самого. Вид конечно в административном интерфейсе непригляден, и мы попробуем это исправить. Добавим в `bugtracker/admin.py` новый класс, который определяет интерфейс списка тикетов в админке и зарегистрируем его:

```
class TicketAdmin(admin.ModelAdmin):
    list_display = ('closed', 'title', 'text', 'created', 'user')
    list_filter = ['created', 'closed']
    search_fields = ['title', 'text']

admin.site.register(Ticket, TicketAdmin)
```

После этого админка станет намного удобнее. Можно отметить пункт «Редактирование багрепортов» как выполненный.

Следующее, что мы сделаем — добавим возможность просмотра списка тикетов, не заходя в административный интерфейс. Для этого нам нужно создать базовый `html`-шаблон и шаблон собственно списка тикетов. Базовый шаблон будет находиться в файле `templates/base.html`:

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Bugtracker - {% block title_block %}{% endblock %}</title>
  </head>
  <body>
    {% block content_block %}
    {% endblock %}
  </body>
</html>
```

Как видите он очень прост и содержит в себе два блока, для заголовка и содержимого.

Шаблон списка создадим в файле `templates/list.html`

```
{% extends 'base.html' %}
{% block title_block %}Main{% endblock %}
{% block content_block %}
<div>
  <h1>Bug list</h1>
  <table>
    <thead>
      <tr>
        <td>Title</td>
        <td>Created</td>
        <td>Author</td>
        <td>Status</td>
      </tr>
    </thead>
    <tbody>
      {% for ticket in object_list %}
      <tr>
        <td>{{ ticket.title }}</td>
```

```

        <td>{{ ticket.created|date }}</td>
        <td>{{ ticket.user }}</td>
        <td>{{ ticket.closed|yesno:"CLOSED, OPENED" }}</td>
    </tr>
    {% empty %}
    <tr>
        <td colspan=4>No tickets yet.</td>
    </tr>
    {% endfor %}
</tbody>
</table>
</div>
{% endblock %}

```

Все это будет подставлено в базовый шаблон вместо

```

{% block content_block %}
{% endblock %}

```

. Содержимое представляет собой таблицу из четырех колонок, которая будет заполнена данными тикетов, а при их отсутствии (пустая) в таблице будет надпись

```
No tickets yet.
```

Теперь напишем class-based view для этой функции в файле `bugtracker/views.py`, он очень прост:

```

from django.views.generic import ListView
from .models import Ticket

class BugListView(ListView):
    model = Ticket
    template_name = 'list.html'

```

Здесь определена модель для отображения в виде списка, и шаблон для отображения.

Осталось только, создать новый паттерн для URL `"/bugs/"` в файле `bugtracker/urls.py`: добавив

```
from .views import BugListView
```

и

```
url(r'^$', BugListView.as_view(), name='index'),
```

(как мы помним, в файле `project/urls.py` — общего для всего проекта, мы определили, что паттерны для `"/bugs/"` находятся в `bugtracker/urls.py`, соответственно регэксп паттерна для URL `"/bugs/"` будет выглядеть именно так: `r'^$',`

Заходим на <http://127.0.0.1:8000/bugs/> и видим страшноватую таблицу со списком всех наших тикетов. Так как дизайн мы прикрутим потом, можем смело поставить статус «Closed» для этого этапа в админке.

Этап №2 — детали тикета

Выполняем:

```
git checkout -f part-2
```

Добавим возможность просмотра каждого тега по-отдельности, используя `DetailView`, действуем по старой схеме.

Создаем шаблон `templates/detail.html`:

```

{% extends 'base.html' %}
{% block title_block %}{{ object.title }}{% endblock %}
{% block content_block %}
<div>
    <h1>{{ object.closed|yesno:"CLOSED, OPENED" }} - {{ object.title }}</h1>
    <p>{{ object.user }} - {{ object.created|date }}</p>
    <p>{{ object.text }}</p>
</div>
{% endblock %}

```

Создаем view в `bugtracker/views.py`:

```
from django.views.generic import DetailView

class BugDetailView(DetailView):
    model = Ticket
    template_name = 'detail.html'
```

и в `bugtracker/urls.py`:

```
from .views import BugDetailView
```

в `urlpatterns` добавляем

```
url(r'^(?P<pk>[0-9]+)/$', BugDetailView.as_view(), name='detail'),
```

Кроме того, добавляем в шаблон `templates/list.html` ссылку на такой URL для каждого тикета, можно например сделать так:

```
<td><a href="{% url 'detail' ticket.pk %}">{{ ticket.title }}</a></td>
```

благодаря url `'detail'` наши урлы будут генерироваться автоматически, и даже если структура урлов поменяется, то шаблон не «поломаются»

Проверяем, перейдя на <http://127.0.0.1:8000/bugs/> и затем кликнув по любой из ссылок. Если все работает, отмечаем в админке, что тикет закрыт.

Этап №3 — регистрация пользователей

```
git checkout -f part-3
```

Правильнее, конечно, вынести работу с пользователями в отдельное приложение и воспользоваться, например, [django-registration](#) для удобной работы с активацией, сменой и восстановлением паролей и т.д., но так как мы обучаемся, то не будем использовать батарейки, дабы не умножать сущности сверх необходимого.

При регистрации пользователей мы будем использовать встроенные функции django по работе с формами, для этого создадим шаблон `templates/register.html`:

```
{% extends 'base.html' %}
{% block title_block %}Registration{% endblock %}
{% block content_block %}
{% if user.is_authenticated %}
    <a href="{% url 'index' %}">Back to main page</a>
{% else %}
    <form action="{% url 'register' %}" method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <input type="submit" value="Submit" />
    </form>
{% endif %}
{% endblock %}
```

В случае, если юзер не залогинен, то мы отображаем форму, которые передается в шаблон из следующего view (`bugtracker/views.py`):

```
from django.views.generic import CreateView
from django.core.urlresolvers import reverse_lazy
from django.contrib.auth.forms import UserCreationForm

class RegisterView(CreateView):
    form_class = UserCreationForm
    template_name = 'register.html'
    success_url = reverse_lazy('index')
```

success_url — это куда пользователя будет направлять при успешной регистрации, урл будет излечен из `urlpatterns` с помощью `reverse_lazy('index')`, то есть в нашем случае `"/bugs/"`

Дополним базовый шаблон ссылкой на форму регистрации, которая будет отображаться только если юзер не залогинен:

```
{% if user.is_authenticated %}
    <div>Welcome, {{ user.username }}!</div>
{% else %}
    <div><a href="{% url 'register' %}">Register</a></div>
```

```
{% endif %}
```

Не забываем добавить импорт `RegisterView` в `bugtracker/urls.py` и в `urlpatterns` — `url(r'^register/$', RegisterView.as_view(), name='register')`,
Теперь можно проверить работоспособность и перейти к следующему этапу.

Этап №4 — логин и логат

```
git checkout -f part-4
```

Здесь все просто, мы будем использовать встроенные джанговские шорткаты. Для логата шаблон, не понадобится, так как отображать-то в этой функции и нечего, нас просто будет переносить на главную страницу. Поэтому создаем шаблон только для логина (`templates/login.html`):

```
{% extends 'base.html' %}
{% block title_block %}Login{% endblock %}
{% block content_block %}
<form action="{% url 'login' %}" method="post">
  {% csrf_token %}
  {% if form.non_field_errors %}
    <p>
      {% for error in form.non_field_errors %}
        {{ error }}
      {% endfor %}
    </p>
  {% endif %}
  {% for field in form %}
    <div>
      {{ field.label_tag }}
      {{ field }}
      {% if field.errors %}
        <p>
          {% for error in field.errors %}
            {{ error }}
          {% endfor %}
        </p>
      {% endif %}
    </div>
  {% endfor %}
  <input type="submit" value="Submit" />
</form>
{% endblock %}
```

В `bugtracker/urls.py` пропишем следующее:

```
from django.core.urlresolvers import reverse_lazy
```

и

```
url(r'^login/$', 'django.contrib.auth.views.login',
    {"template_name" : "login.html"}, name="login"),
url(r'^logout/$', 'django.contrib.auth.views.logout',
    {"next_page" : reverse_lazy('login')}, name="logout"),
```

А `templates/base.html` переделаем так:

```
{% if user.is_authenticated %}
  <div>Welcome, {{ user.username }}
  | <a href="{% url 'logout' %}">Logout</a></div>
{% else %}
  <div><a href="{% url 'register' %}">Register
  | <a href="{% url 'login' %}">Login</a></div>
{% endif %}
```

Что бы после логина нас редиректило на главную страницу в settings.py пропишем

```
from django.core.urlresolvers import reverse_lazy
LOGIN_REDIRECT_URL = reverse_lazy('index')
```

Поиграем вдоволь, создавая новых пользователей, логинаясь и разлогиниваясь под ними.

Этап №5 — добавление нового тикета

```
git checkout -f part-5
```

Шаблон templates/add.html очень похож на тот, что мы создавали для RegisterView, только наоборот — функционал доступен авторизованному пользователю:

```
{% extends 'base.html' %}
{% block title_block %}Add ticket{% endblock %}
{% block content_block %}
{% if user.is_authenticated %}
    <a href="{% url 'index' %}">Back to main page</a>
    <form action="{% url 'add' %}" method="post">
        {{ csrf_token }}
        {{ form.as_p }}
        <input type="submit" value="Submit" />
    </form>
{% else %}
    <p>You should be logged in to add tickets!</p>
{% endif %}
{% endblock %}
```

Добавляем куда-нибудь в базовый шаблон ссылочку на add:

```
<div><a href="{% url 'add' %}">Add ticket</a></div>
```

View будет таким,

```
class BugCreateView(CreateView):
    model = Ticket
    template_name = 'add.html'
    fields = ['title', 'text']
    success_url = reverse_lazy('index')

    def form_valid(self, form):
        form.instance.user = self.request.user
        return super(BugCreateView, self).form_valid(form)
```

То есть пользователю будут видны только поля title и text, а благодаря переопределению метода form_valid в поле user будет подставлен индекс текущего пользователя.

Прописываем паттерн для urlа 'add/' в bugtracker/urls.py, не забыв импортировать BugCreateView:

```
url(r'^add/$', BugCreateView.as_view(), name='add'),
```

Попробуем добавить новых тикетов, и переходим к следующему этапу.

Этап №6 — сделать нарядно

```
git checkout -f part-6
```

Все это более-менее удовлетворительно работает, но выглядит страшновато. На мой взгляд, самый простой и удобный способ улучшить наш интерфейс, доступный даже неспециалисту в верстке (как я) — это использовать twitter bootstrap. Для этого скачаем соответствующую библиотеку и распакуем её в папку static. После чего мы приступим к верстке шаблонов. У bootstrap куча примеров,

поэтому особых сложностей возникнуть не должно. Поскольку, приводить здесь простыни получившегося HTML не имеет смысла, а нюансы верстки под bootstrap выходят за рамки этой статьи (и моих познаний), просто посмотрите исходники в [репозитории](#), или выполните `git checkout -f final`, что бы получить окончательный вариант кода.

Стоит заметить, что третья версия bootstrap требует добавлять `class=«form-control»` к инпутам в формах. Известный мне самый простой метод, сделать это в шаблонах django (не используя дополнительных батареек) — это кастомный фильтр, который описан [здесь](#). Буду благодарен, если более искушенные читатели моей статьи поделятся своим опытом в комментариях.

Итог

Буквально за пару часов, а то и меньше, мы с вами создали вполне функциональное приложение, которое при дальнейшем улучшении, настройке и развитии, может хорошо послужить, в каких то других проектах, а может быть и как standalone-сервис. Вот мои несколько вполне реализуемых идей для его улучшения:

- HTTP API (сразу появляется возможность взаимодействия практически со всем чем угодно — программы, консольные скрипты, другие веб-сервисы и т.д.
- Побольше статусов для тикетов
- Пагинация в шаблоне вывода списка
- Возможность поручить исполнение тикета другому юзеру и контролировать статус
- Загрузка файлов и изображений
- Легкая настройка и кастомизация дополнительных полей для тикетов
- Система плагинов

Жду ваших замечаний, идей и предложений в комментариях. Спасибо!

django, python, bugtracking, практика

—

426

15

svfat 12,6 G+

А как ты используешь новые
измерения в создании приложений?

Комментарии (0) отслеживать новые: ☐ в почте ☐ в треке

Веб-дизайнер

php-разработчик

UI/UX дизайнер

Remote Scala Engineer

Front-End AngularJS Engineer

iOS или Android-разработчик

Sr. PHP / Drupal Developer

Web программист на Дياسофт Фронт

Ведущий программист SQL

Java Web Developer/ Senior Developer (GWT, GAE)

все вакансии

Front-End разработчик для полного редизайна сайта по скидкам

Для сайта посвященного медицине и здоровью автор

HTML/JS вёрстка сайта

Автоматизировать процессы на на 6 сайтах работы

Пропали все сайты с VDS сервера

Создание ТЗ под промосайт компании дистрибуция напитков

Небольшие изменения в верстке лендинга

Фронтэнд разработка и верстка сайта

Создание промо сайта для FMCG компании (напитки)

Аналитика По бесконтактным платежам

все заказы

