LESSON 6
URLs and Slugs

Topics Covered

☑ Slugs.

☑ Utility functions.

☑ Overriding `models.Model.save()`.

Introduction

In this lesson, you will learn to create slugs, rather than primary keys, to use in URLs. In doing so, you will learn to organize and write your own utility functions and to override the model's `save()` method.

6.1. Slugs

A *slug* is a string used in a URL to uniquely identify a web page. For example, instead of:

```
http://127.0.0.1:8000/jokes/joke/1/
```

…the URL to the page with the joke about the chicken crossing the road could be:

```
http://127.0.0.1:8000/jokes/joke/why-did-the-chicken-cross-the-road/
```

This provides several benefits over the primary-key approach:

1. Primary keys shouldn't be exposed as hackers can use that data to try to mess with the database.
2. When you use primary keys as part of the URL, if a joke's primary key changes, which can happen if a joke is deleted and then re-added, the URL of the joke will also change. This means that existing links to that joke from other pages will get broken. If you use slugs, changes in the primary key will not affect changes in the URL (as long as you keep the slug the same).
3. Some search engines use the URL string to learn what the page is about. "why-did-the-chicken-cross-the-road" is much more meaningful than "1".

❖ 6.1.1. `SlugField`

Django models include a `SlugField` field type, which allows for strings that contain letters, numbers, underscores, and hyphens. By default, the maximum length of the string is 50, but that can be changed with the `max_length` argument.

❖ 6.1.2. Implementing Slugs

The steps involved for implementing slugs are:

1. Create a function that automatically generates a unique slug for a model.
2. Add a `slug` field to the model.
3. Use the function created above to auto-populate the `slug` field when a record is saved.
4. Modify the URL pattern to use the slug.

If you had started with this approach, that would be it, but as you already have data in a **jokes** table, you need to do a couple of additional things. Normally, you won't allow `null` values for slugs. However, records that already exist in the table won't have a value for the new `slug` field. In some cases, when you need to add a new required field to a model, you can set a default value for the field. All existing records will get that default value. However, if the field must be unique, as is the case with slugs, setting a default value won't work.

The solution is as follows:

1. Allow for `null` values when first adding the field.
2. Create unique slugs for all existing records.
3. Change the model to disallow `null` values.

You will do all of this in the following exercises.

Exercise 27: Creating a Slug-generating Function

15 to 20 minutes

As you are likely to use slugs in more than one app, it makes sense to create the slug-generating function in the `common` app. Functions like this are called `utility` functions, and as your project may end up having a lot of them, it is important to organize them well:

1. Within the `djangojokes.com/common` folder, create a `utils` folder. This will be the home for all your project's general utility functions.
2. Within the new `utils` folder, create a file called `text.py` for utilities that work with text and add the following content:[47]

47. **Don't want to type?** Copy from `starter-code/urls-and-slugs/text.py`.

Exercise Code 27.1: common/utils/text.py

```
1.   import random
2.   import string
3.
4.   from django.utils.text import slugify
5.
6.   def unique_slug(s, model, num_chars=50):
7.       """
8.       Return slug of num_chars length unique to model
9.
10.      `s` is the string to turn into a slug
11.      `model` is the model we need to use to check for uniqueness
12.      """
13.      slug = slugify(s)
14.      slug = slug[:num_chars].strip('-')
15.      while True:
16.          dup = model.objects.filter(slug=slug)
17.          if not dup:
18.              return slug
19.
20.          slug = slug[:39] + '-' + random_string(10)
21.
22.
23.  def random_string(num_chars=10):
24.      letters = string.ascii_lowercase
25.      return ''.join(random.choice(letters) for i in range(num_chars))
```

Code Explanation

Read through this code to make sure it makes sense to you. **Some things to notice:**

1. You import `slugify` from Django's own text utility functions at `django.utils.text`. The `slugify()` function does the following:
    A. Converts to ASCII.
    B. Converts spaces to hyphens.
    C. Removes characters that aren't alphanumerics, underscores, or hyphens.
    D. Converts to lowercase.
    E. Strips leading and trailing whitespace.

2. In the `unique_slug()` function, you first slugify, slice (to make sure the string isn't too long), and strip off any hyphens (so the slug doesn't end with a hyphen). Then, you make sure the slug is unique for the passed-in model:

```
while True:
    dup = model.objects.filter(slug=slug)
    if not dup:
        return slug

    slug = slug[:39] + '-' + random_string(10)
```

`model.objects.filter(slug=slug)` returns any existing records with that slug.[48]

   A. If it doesn't find any records with the generated slug, it will return it.
   B. If it does find a record with the generated slug, it will create a new slug using the first 39 characters of the slug followed by a hyphen and a random string of ten lowercase ASCII characters generated by the `random_string()` function. It will repeat that process until it comes up with a unique slug at which point it will return the slug.

Git Commit

Commit your code to Git.

48. We will cover querying models in detail in the Making Queries, Ajax, and View Functions lesson (see page 407).