To save a base64 encoded image in a Django `FileField`, you need to decode the base64 data and create a file object from it. Here's a step-by-step guide on how to achieve this:

1. **Decode Base64 Data:**

   Decode the base64 encoded image data to get the binary image content. You can use the `base64` module in Python for this.

2. **Create a File Object:**

   After decoding the base64 data, create a `ContentFile` object from the binary content. The `ContentFile` class is provided by Django and allows you to create a file-like object from content in memory.

3. **Save to FileField:**

   Assign the `ContentFile` object to your model's `FileField`.

Here's an example implementation within a Django model:

```
from django.db import models
from django.core.files.base import ContentFile
import base64

class ImageModel(models.Model):
    image = models.FileField(upload_to='images/')

    @classmethod
    def create_from_base64(cls, base64_data):
        image_data = base64.b64decode(base64_data)
        image_content = ContentFile(image_data, name='image.png')
        image_model = cls.objects.create(image=image_content)
        return image_model
```

In this example, `ImageModel` is a Django model with a `FileField` named `image`. The `create_from_base64` method takes a base64 encoded image data as input, decodes it, creates a `ContentFile`, and then creates an instance of the model with the decoded image content.

You can use this method like this:

```
base64_data = "..."  # Replace with your base64 encoded image data
image_model = ImageModel.create_from_base64(base64_data)
```

Make sure to replace `"..."` with the actual base64 encoded image data.

Remember to adjust the model's `upload_to` path and other settings as needed for your project.

Also, keep in mind that handling large base64 encoded images can consume a significant amount of memory. It's generally recommended to process and save large files directly using a file upload mechanism instead of embedding them as base64 data.