To save a base64-encoded image in a Django `FileField` using a custom approach, you can follow these steps:

1. Create a custom file field that accepts base64 data and converts it into a file.
2. Override the `to_python` method of the custom field to handle base64 data and create a file object.
3. Use the custom file field in your model to store the images.

Here's an example implementation:

1. Create a custom file field:

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

class Base64FileField(models.FileField):
    def to_python(self, value):
        if value is None:
            return None

        if isinstance(value, str) and value.startswith('data:'):
            # Extract file type and base64 data from the input
            header, data = value.split(';base64,')
            file_extension = header.split('/')[-1]

            # Convert base64 data to bytes
            file_data = base64.b64decode(data)

            # Create a ContentFile object from bytes
            return ContentFile(file_data, name='temp.' + file_extension)

        return super().to_python(value)
```

2. Use the custom file field in your model:

```
class YourModel(models.Model):
    image = Base64FileField(upload_to='images/')
```

3. In your view or serializer, you can now assign a base64-encoded image to the `image` field of your model:

```
base64_image = "..."
your_model_instance = YourModel()
```

```
your_model_instance.image = base64_image
your_model_instance.save()
```

Remember to adjust the code according to your needs and ensure proper validation and error handling.

Please note that this approach may not handle all possible edge cases, and you might need to enhance it based on your specific requirements. Additionally, ensure that you have proper security measures in place, such as validation and user authorization, to prevent potential security vulnerabilities.