

Task 24 - Django "sticky_notes" App

Design Diagrams "sticky_notes" project

Helder Paixao - "HP24010013265"

3rd June, 2024



Introduction

1. **Use Case Diagram**
2. **Sequence Diagram**
3. **Class Diagram**
4. **User Access Table Diagram**

When reading this pdf.file, view it in “pageless” mode: The pageless format allows you to add wide images and tables, and consume content without the interruption of page breaks.

Use Case Diagram

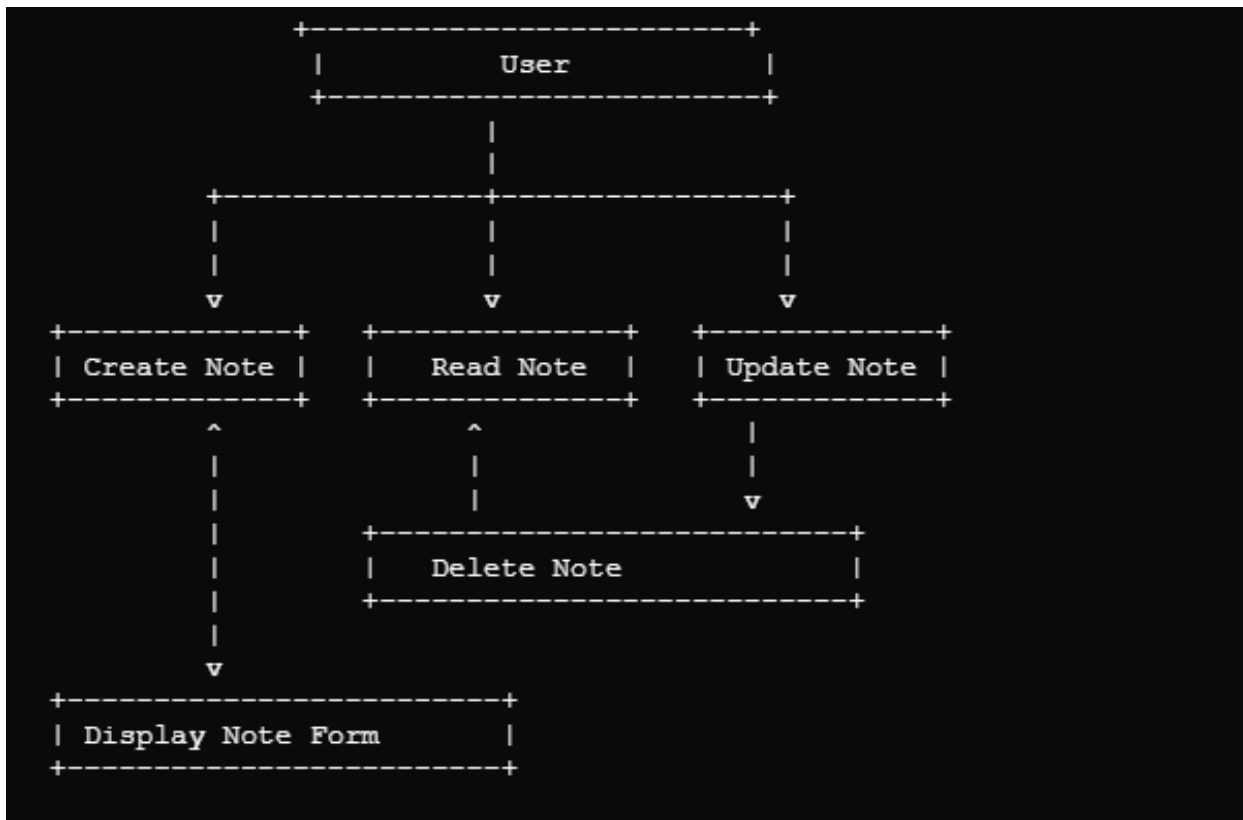
Actors:

- User: The primary actor who interacts with the sticky notes application.

Use Cases:

- Create Note: The user can create a new sticky note.
- Read Note: The user can view a list of notes and read the details of a specific note.
- Update Note: The user can edit the details of an existing note.
- Delete Note: The user can delete an existing note.

Diagram:



Detailed Breakdown

1. Create Note:
 - Description: The user creates a new note by providing the title and content.
 - Interaction: User fills out and submits a form to create a note.
2. Read Note:
 - Description: The user views a list of all notes and can read the details of each note.
 - Interaction: User clicks on a note to view its details.
3. Update Note:
 - Description: The user edits the title or content of an existing note.
 - Interaction: User submits a form to update the note's details.
4. Delete Note:
 - Description: The user deletes an existing note.
 - Interaction: User confirms the deletion of a note.
5. Display Note Form:

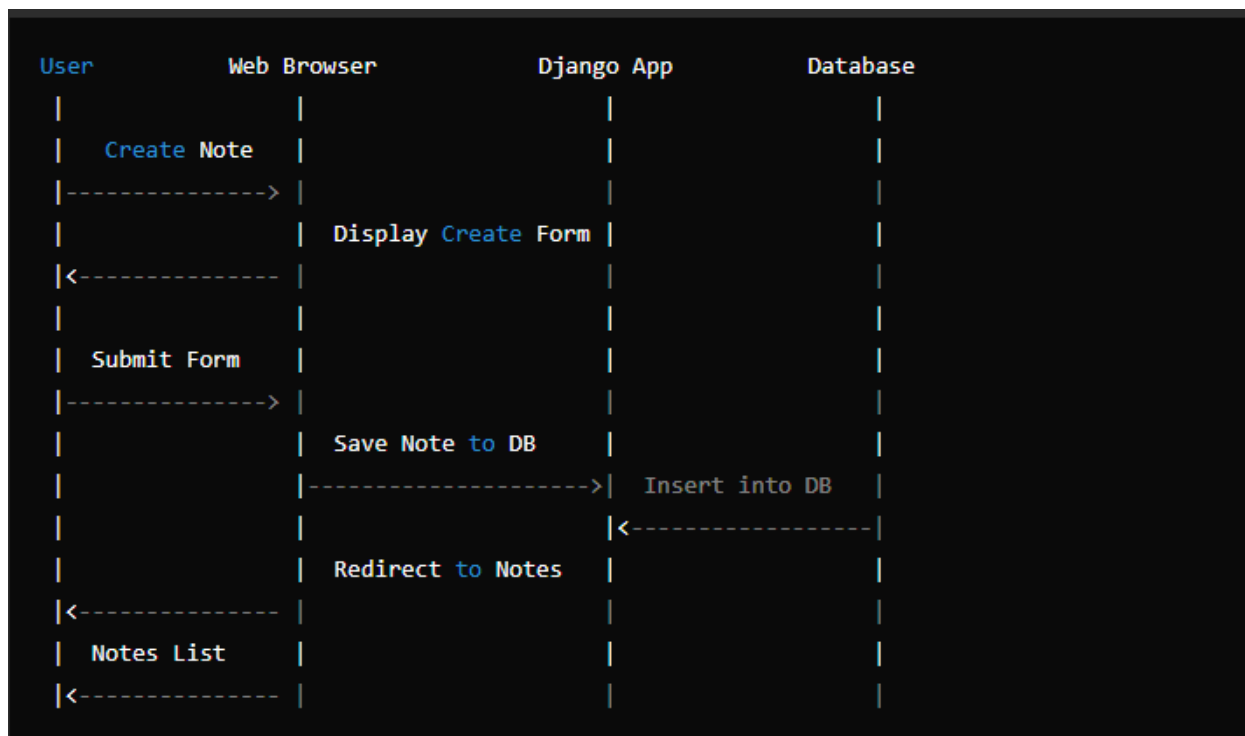
- Description: A form is displayed for creating or updating a note.
- Interaction: Form is used in both the create and update use cases.

The Use Case Diagram provides a high-level overview of how the user interacts with the "sticky_notes" Django project. Each use case represents a core functionality that the application provides, facilitating the creation, reading, updating, and deletion of sticky notes.

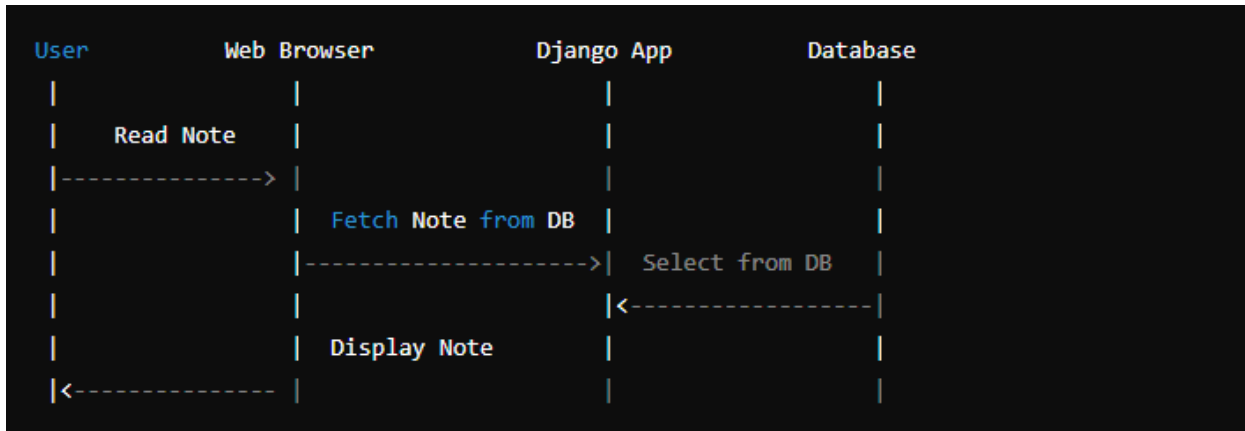
Sequence Diagram

The sequence diagram shows the flow of interactions between the user and the system for each use case.

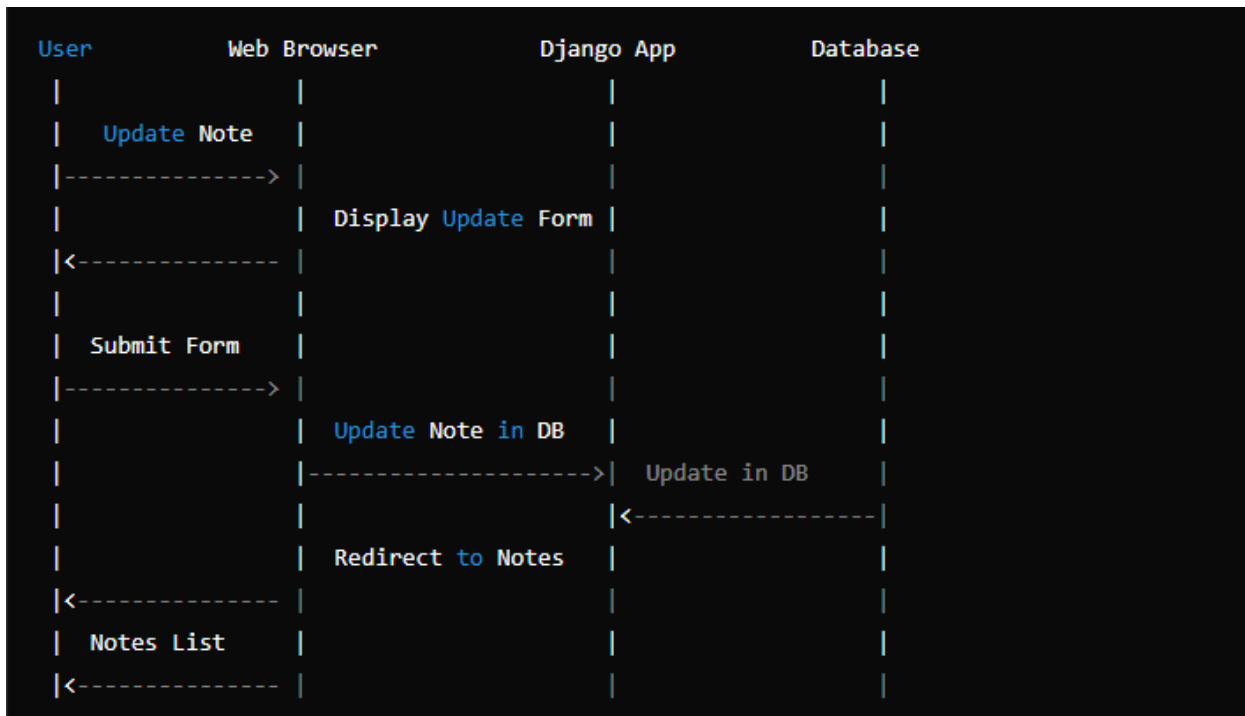
Create Note Sequence Diagram



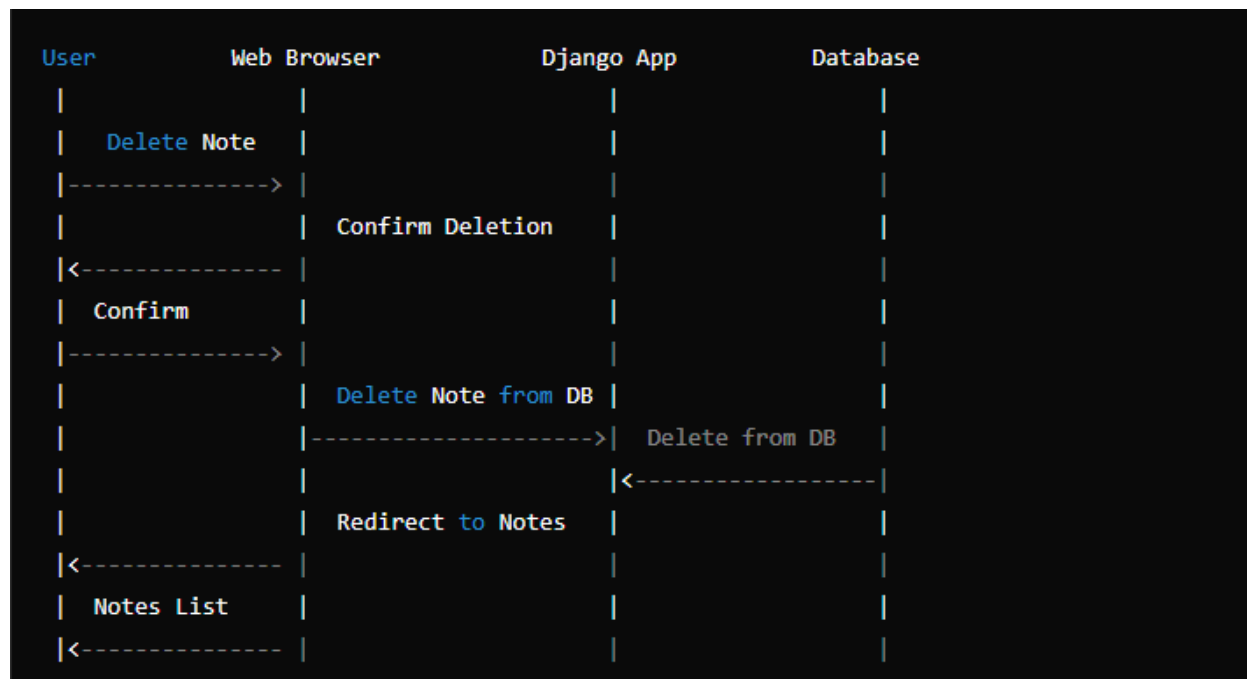
Read Note Sequence Diagram



Update Note Sequence Diagram



Delete Note Sequence Diagram



Class Diagram

Diagram:

```
+-----+
|      Note      |
+-----+
| - id: Integer   |
| - title: CharField |
| - content: TextField |
| - created_at: DateTime |
+-----+
| + save()        |
| + delete()      |
+-----+
```

```
      ^
      |
      | 1
      |
      |
1|
v
```

```
+-----+
|     NoteForm    |
+-----+
| - Meta: ModelForm.Meta |
+-----+
| + is_valid(): bool   |
| + save()            |
+-----+
```

```
      ^
      |
      | 1
      |
      |
1|
v
```

```
+-----+
|      Views      |
+-----+
| + note_list(request) |
| + note_detail(request, |
|   pk)              |
| + note_create(request) |
| + note_update(request, |
|   pk)              |
| + note_delete(request, |
|   pk)              |
+-----+
```

```

      ^
      |
      | 1
      |
      |
1|
      v

+-----+
|      URLConfig      |
+-----+
| + urlpatterns: list  |
+-----+
| + path()             |
| + include()          |
+-----+

      ^
      |
      | 1
      |
      |
1|
      v

+-----+
|      Templates      |
+-----+
| + base.html          |
| + note_list.html     |
| + note_detail.html   |
| + note_form.html     |
| + note_confirm_delete.html |
+-----+

```

Detailed Breakdown

1. Note Model:

- Represents the structure of a note in the database.
- Fields:
 - `id`: Primary key, automatically added by Django.

- `title`: CharField for the title of the note.
- `content`: TextField for the note content.
- `created_at`: DateTimeField automatically set when a note is created.

2. NoteForm:

- A Django form used to create and update notes.
- Based on the `Note` model.
- Contains methods for form validation (`is_valid()`) and saving data (`save()`).

3. Views:

- Functions to handle CRUD operations for notes.
- `note_list(request)`: Displays a list of all notes.
- `note_detail(request, pk)`: Displays the details of a single note.
- `note_create(request)`: Handles the creation of a new note.
- `note_update(request, pk)`: Handles the updating of an existing note.
- `note_delete(request, pk)`: Handles the deletion of a note.

4. URLConfig:

- Defines URL patterns and maps them to views.
- `urlpatterns`: A list of URL patterns.
- `path()`: Function to define individual URL patterns.
- `include()`: Function to include other URL configurations.

5. Templates:

- HTML templates used to render the views.
- `base.html`: Base template extended by other templates.
- `note_list.html`: Template for displaying the list of notes.
- `note_detail.html`: Template for displaying note details.
- `note_form.html`: Template for creating and updating notes.
- `note_confirm_delete.html`: Template for confirming note deletion.

This class diagram provides a comprehensive overview of the components and their interactions within the "sticky_notes" Django project. Each class represents a key part of the Django framework, illustrating the Model-View-Template (MVT) architecture.

User Access Table Diagram

The table includes:

- User Types: Types of users that interact with the application (e.g., Admin, Registered User, Guest).
- CRUD Operations: Permissions for Create, Read, Update, and Delete operations on the sticky notes.

Table Structure

User Type	Create	Read	Update	Delete
Admin	✓	✓	✓	✓
Registered User	✓	✓	✓	✓
Guest		✓		

Detailed Breakdown

1. Admin:
 - Create: Can create new sticky notes.
 - Read: Can read/view all sticky notes.
 - Update: Can update any sticky note.
 - Delete: Can delete any sticky note.
2. Registered User:
 - Create: Can create new sticky notes.
 - Read: Can read/view all sticky notes.
 - Update: Can update their own sticky notes.
 - Delete: Can delete their own sticky notes.
3. Guest:

- Create: Cannot create new sticky notes.
- Read: Can read/view all sticky notes.
- Update: Cannot update any sticky notes.
- Delete: Cannot delete any sticky notes.

Diagram

User Type	Create	Read	Update	Delete
Admin	✓	✓	✓	✓
Registered User	✓	✓	✓	✓
Guest		✓		

Integration with Django

In Django, user access control can be managed using permissions and user groups. Here's how this can be integrated into the project:

1. Admin: This user type is typically created using Django's admin interface. Admin users have all permissions.
2. Registered User: These users are authenticated users who can be given specific permissions to create, read, update, and delete their own notes.
3. Guest: These users are not authenticated and typically have limited access, such as only being able to read notes.

Implementation Example

You can implement this in your Django project using the `permissions` framework:

Model Example:

```
from django.db import models
from django.contrib.auth.models import User

class Note(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE)

    class Meta:
        permissions = [
            ("can_read_note", "Can read note"),
            ("can_create_note", "Can create note"),
            ("can_update_note", "Can update note"),
            ("can_delete_note", "Can delete note"),
        ]
```

Example:

```
from django.contrib.auth.decorators import permission_required
from django.shortcuts import render

@permission_required('app_name.can_create_note')
def create_note_view(request):
    # logic for creating a note

@permission_required('app_name.can_read_note')
def read_note_view(request):
    # logic for reading a note

@permission_required('app_name.can_update_note')
def update_note_view(request):
    # logic for updating a note

@permission_required('app_name.can_delete_note')
def delete_note_view(request):
    # logic for deleting a note
```

This setup ensures that only authorized users can perform specific actions, enhancing the application's security and usability.
