1. Database Schema and Setup Scripts

Database Schema Design

Users Table (users):

- id: Auto-incrementing primary key
- username: Unique identifier for each user
- password: Encrypted password for authentication
- email: User's email address (must be unique)

Posts Table (posts):

- id: Auto-incrementing primary key
- title: Title of the blog post
- content: Full content of the blog post
- author_id: Foreign key referencing the users table (author of the post)
- created_at: Timestamp when the post was created
- updated_at: Timestamp when the post was last updated

Comments Table (comments):

- id: Auto-incrementing primary key
- post_id: Foreign key referencing the posts table (which post the comment belongs to)
- content: The content of the comment
- author_id: Foreign key referencing the users table (author of the comment)
- created_at: Timestamp when the comment was created

2. Comprehensive API Documentation

The API has multiple endpoints to manage users, blog posts, and comments. Below are the API endpoints and details for each.

Authentication Endpoints

1. **Register a New User POST METHOD**
   Endpoint: http://127.0.0.1:8000/api/register/
   Request Body: {

   "username": "newuser",

   "password": "password123",

"password2": "password123",

"email": "newuser@example.com" }

Response: 201 Created

{

"message": "User created successfully"

}

2. **Login POST METHOD**
   Endpoint: http://127.0.0.1:8000/api/login/
   Request Body: {
   "username": "newuser",
   "password": "password123" }

   Response: 200 OK

   {

   "access": "jwt_access_token",

   "refresh": "jwt_refresh_token"}

Blog Post Endpoints

1. **Create a Post POST METHOD**
   Endpoint: http://127.0.0.1:8000/api/posts/
   Request Body:{

   "title": "Sample Post",

   "content": "This is the content of the post",

   "author": 1

   }

   Response: 201 Created

   {

   "id": 1,

   "title": "Sample Post",

   "content": "This is the content of the post",

   "author": 1,

   "created_at": "2023-08-15T12:34:56Z",

   "updated_at": "2023-08-15T12:34:56Z"

   }

2. **Get All Posts GET METHOD**
   Endpoint: http://127.0.0.1:8000/api/posts/
   Response: 200 OK
   [
     {
       "id": 1,
       "title": "Sample Post",
       "content": "This is the content of the post",
       "author": 1,
       "created_at": "2023-08-15T12:34:56Z",
       "updated_at": "2023-08-15T12:34:56Z"
     }
   ]

3. **Get a Single Post GET METHOD**
   Endpoint: http://127.0.0.1:8000/api/posts/{id}
   Response: 200 OK

   { "id": 1,

   "title": "Sample Post",

   "content": "This is the content of the post",

   "author": 1,

   "created_at": "2023-08-15T12:34:56Z",

   "updated_at": "2023-08-15T12:34:56Z"

   }

4. **Update a Post PUT METHOD**
   Endpoint: http://127.0.0.1:8000/api/posts/{id}
   Request Body: {
     "title": "Updated Post Title",
     "content": "Updated content"
   }
   Response: 200 OK

5. **Delete a Post DELETE METHOD**
   Endpoint: http://127.0.0.1:8000/api/posts/{id}
   Response: 204 No Content

Comment Endpoints

1. **Create a Comment POST METHOD**
   Endpoint: http://127.0.0.1:8000/api/comments/
   Request Body: {

   "content": "This is a comment",

"post": 1,

"author": 1}

Response: 201 Created

2. **Get Comments for a Post GET METHOD**
   Endpoint: http://127.0.0.1:8000/api/comments/{post_id}
   Response: 200 OK
   [
     {
       "id": 1,
       "content": "This is a comment",
       "post": 1,
       "author": 1,
       "created_at": "2023-08-15T12:34:56Z"
     }
   ]

3. **Update a Comment PUT METHOD**
   Endpoint: http://127.0.0.1:8000/api/comments/{id}
   Request Body: {
     "content": "Updated comment content"
   }
   Response: 200 OK

4. **Delete a Comment DELETE METHOD**
   Endpoint: http://127.0.0.1:8000/api/comments/{id}
   Response: 204 No Content

Unit and Integration Test Cases

Unit Test Cases for Post

```
from django.urls import reverse
from rest_framework import status
from rest_framework.test import APITestCase
from django.contrib.auth.models import User
from blog.models import Post

class PostTests(APITestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='testuser',
password='testpassword')
        self.client.login(username='testuser', password='testpassword')

    def test_create_post(self):
        url = reverse('post-list')
        data = {'title': 'Test Post', 'content': 'Test content', 'author': self.user.id}
```

```
        response = self.client.post(url, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        self.assertEqual(Post.objects.count(), 1)
        self.assertEqual(Post.objects.get().title, 'Test Post')

    def test_get_posts(self):
        Post.objects.create(title='Test Post', content='Test content', author=self.user)
        url = reverse('post-list')
        response = self.client.get(url, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data), 1)
```

Unit Test Cases for Comment

```
from rest_framework import status
from rest_framework.test import APITestCase
from django.contrib.auth.models import User
from blog.models import Post, Comment

class CommentTests(APITestCase):
    def setUp(self):
        self.user = User.objects.create_user(username='testuser',
password='testpassword')
        self.post = Post.objects.create(title='Test Post', content='Test content',
author=self.user)
        self.client.login(username='testuser', password='testpassword')

    def test_create_comment(self):
        url = reverse('comment-list')
        data = {'content': 'Test comment', 'post': self.post.id, 'author': self.user.id}
        response = self.client.post(url, data, format='json')
        self.assertEqual(response.status_code, status.HTTP_201_CREATED)
        self.assertEqual(Comment.objects.count(), 1)
        self.assertEqual(Comment.objects.get().content, 'Test comment')

    def test_get_comments(self):
        Comment.objects.create(content='Test comment', post=self.post,
author=self.user)
        url = reverse('comment-list') + '?post_id=' + str(self.post.id)
        response = self.client.get(url, format='json')
        self.assertEqual(response.status_code, status.HTTP_200_OK)
        self.assertEqual(len(response.data), 1)
```

Conclusion:

- The database schema is defined using three core tables: users, posts, and comments.

- The API documentation covers endpoints for user registration, authentication, blog posts, and comments, providing comprehensive details about requests and responses.
- Unit and integration tests ensure the API functions as expected for creating, reading, updating