Linting results

All files were linted with CI Python Linter

Project files

Files in the *basilandthyme directory*

basilandthyme/settings.py

code institute

CI Python Linter

```python
"""
Django settings for basilandthyme project.
Generated by 'django-admin startproject' using Django 4.2.10.
"""

from pathlib import Path
import os
import sys
import dj_database_url
import cloudinary
if os.path.isfile('env.py'):
    import env


BASE_DIR = Path(__file__).resolve().parent.parent
TEMPLATES_DIR = os.path.join(BASE_DIR, 'templates')


SECRET_KEY = os.environ.get("SECRET_KEY")


DEBUG = True

ALLOWED_HOSTS = ['.herokuapp.com', '127.0.0.1']


INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
```

Settings:

Results:

All clear, no errors found

basilandthyme/urls.py

basilandthyme/wsgi.py

code institute

CI Python Linter

```
1  """ WSGI config for basilandthyme project. """
2  import os
3
4  from django.core.wsgi import get_wsgi_application
5
6  os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'basilandthyme.settings')
7
8  application = get_wsgi_application()
9
```

Settings:

Results:

All clear, no errors found

recipe_book

Files in the *recipe_book directory (the app directory)*

recipe_book/admin.py

code institute

CI Python Linter

```python
from django.contrib import admin
from django_summernote.admin import SummernoteModelAdmin
from .models import Recipe, Comment, Favourite, Rating


@admin.register(Recipe)
class RecipeAdmin(SummernoteModelAdmin):
    """
    Customising the Django admin panel for the Recipe model.

    Attributes:
        list_display (tuple): Specifies the fields to display in the list view.
        search_fields (list): Specifies the fields to search on.
        list_filter (tuple): Specifies the fields to filter by.
        prepopulated_fields (dict): Specifies the fields to automatically
            populate based on another field.
        summernote_fields (tuple): A tuple specifying the fields to use
        summernote for text editing.
        fieldsets (list): A list of fieldset configurations for organizing
        fields and adding instructions in the interface.
    """
    list_display = ('title', 'slug', 'status', 'created_on')
    search_fields = ['title', 'content']
    list_filter = ('status', 'created_on', 'category',)
    prepopulated_fields = {'slug': ('title',)}
    summernote_fields = ('content', 'ingredients',)
    fieldsets = [
        (
            "Recipe details",
            {
                "fields": ["title", "slug", "category", "status", "author"],
```

Settings:

Results:

All clear, no errors found

recipe_book/apps.py

code institute

CI Python Linter

```
1  from django.apps import AppConfig
2
3
4  class RecipeBookConfig(AppConfig):
5      default_auto_field = 'django.db.models.BigAutoField'
6      name = 'recipe_book'
7
```

Settings:

Results:

All clear, no errors found

recipe_book/forms.py

code institute

CI Python Linter

```python
from django.utils.translation import gettext_lazy as _
from .models import Comment
from django import forms


class CommentForm(forms.ModelForm):
    """
    Form class for creating and updating Comment instances.

    Attributes:
        model (Model): The model associated with the form.
        fields (tuple): A tuple specifying the fields in the form.
        labels (dict): A dictionary specifying the labels for form fields.
        widgets (dict): A dictionary specifying the widgets to use for form
        fields.

    Credit for infor about widgets:
    https://docs.djangoproject.com/en/5.0/topics/forms/modelforms/#specifying-widgets-to-use-in-the-form-with
        -widgets

    """
    class Meta:
        model = Comment
        fields = ('body',)
        labels = {'body': _('Comment:'), }
        widgets = {
            'body': forms.Textarea(
                attrs={'placeholder': 'Write your comment here...'})
        }

```

Settings:

Results:

All clear, no errors found

recipe_book/urls.py

code institute

CI Python Linter

```python
from . import views
from django.urls import path


urlpatterns = [
    path('', views.FeaturesListView.as_view(), name='home_page'),
    path('recipes/', views.RecipeListView.as_view(), name='recipe_list_page'),
    path(
        'recipes/<slug:slug>/',
        views.RecipeDetailView.as_view(),
        name='recipe_detail'
        ),
    path(
        'favourites/',
        views.FavouritesList.as_view(),
        name='favourites_page'
        ),
    path(
        'add-remove-favourite/',
        views.add_remove_favourite,
        name='add_remove_favourite'
        ),
    path(
        'add-update-rating/',
        views.add_update_rating,
        name='add_update_rating'
        ),
    path('delete-rating/', views.delete_rating, name='delete_rating'),
]

```

Settings:

Results:

All clear, no errors found

Models

Files in the *recipe_book/models directory*

recipe_book/models/__init.py__

recipe_book/models/comment.py

code institute

CI Python Linter

```python
from django.db import models
from django.contrib.auth.models import User
from django.core.validators import MaxLengthValidator
from .recipe import Recipe


class Comment(models.Model):
    """
    Model representing comments on recipes.

    Attributes:
        recipe (ForeignKey to Recipe): The recipe being commented on.
        author (ForeignKey to User): The user who wrote the comment.
        body (TextField): The text content of the comment. Must not exceed 1200
        characters.
        approved (BooleanField): Indicates whether the comment has been
        approved by the site admin. Is True by default.
        created_on (DateTimeField): The date and time of comment creation. Is
        set automatically on comment creation.

    Meta:
        ordering (list of str): Specifies the default ordering, by date and
        time of creation in descending order.

    Methods:
        __str__: Returns a string representation of the comment.
    """
    recipe = models.ForeignKey(
        Recipe, on_delete=models.CASCADE, related_name='comments')
    author = models.ForeignKey(
        User, on_delete=models.CASCADE, related_name='comment_author')
```

Settings:

Results:

All clear, no errors found

recipe_book/models/favourite.py

code institute

CI Python Linter

```python
from django.db import models
from django.contrib.auth.models import User
from .recipe import Recipe


class Favourite(models.Model):
    """
    Model representing the favourites of users for recipes.

    Attributes:
        user (ForeignKey to User): The user who favourited the recipe.
        recipe (ForeignKey to Recipe): The recipe favourited by the user.

    Meta:
        constraints (list of constraints): Ensures that each user can mark a
        recipe as a favourite only once.

    Methods:
        - __str__(): Returns a string representation of the Favourite object.
        - is_recipe_favourite(user, recipe): Checks if a recipe is marked as a
        favourite by a user.
        - is_recipe_favourite_by_ids(user_id, recipe_id): Checks if a recipe is
        marked as a favourite by a user, given user ID and recipe ID.
        - get_user_favourite_ids(user): Retrieves the IDs of recipes favourited
        by a user.
        - create_favourite(user_id, recipe_id): Create a Favourite object given
        a user id and recipe id, given that the favourite object does not
        already exist.
        - delete_favourite(user_id, recipe_id): Delete a Favourite object given
        a user id and recipe id, given that the favourite object exists.
    """
```

Settings:

Results:

All clear, no errors found

recipe_book/models/rating.py

code institute

CI Python Linter

```python
from django.db import models
from django.contrib.auth.models import User
from .recipe import Recipe


class Rating(models.Model):
    """
    Model representing ratings given by users to recipes.

    Attributes:
        - user (ForeignKey to User): The user who rated the recipe.
        - recipe (ForeignKey to Recipe): The recipe being rated.
        - rating (IntegerField): The rating value given by the user, ranging
        from 1 to 5.

    Meta:
        constraints (list of constraints): Ensures that each user can rate a
        recipe only once.

    Methods:
        - __str__(): Returns a string representation of the Rating object.
        - get_recipe_avg_rating(recipe_id): Calculates the average rating for a
        given recipe.
        - get_recipe_no_of_ratings(recipe_id): Gets the number of ratings for a
        given recipe.
        - get_user_rating_of_recipe(user_id, recipe_id): Retrieves the rating
        given by a specific user to a recipe.
    """
    RATING_CHOICES = [
        (1, '1'),
        (2, '2'),
```

Settings:

Results:

All clear, no errors found

recipe_book/models/recipe.py

code institute

CI Python Linter

```python
from django.db import models
from django.core.validators import MinLengthValidator, MaxLengthValidator
from django.contrib.auth.models import User
from cloudinary.models import CloudinaryField


class Recipe(models.Model):
    """
    Model representing a recipe.

    Attributes:
    - title (CharField): The unique title of the recipe. May not be empty and
    must not exceed 70 characters.
    - slug (SlugField): A unique slug for the recipe URL. May not be empty and
    must not exceed 70 characters.
    - author (ForeignKey to User): The user who authored the recipe.
    - feature_image (CloudinaryField): The image of the recipe.
    - alt_text (CharField): Alt-text for the feature image.
    - content (TextField): The detailed content, including recipe instructions.
    Must be 100-5000 characters.
    - ingredients (TextField): Ingredients required for the recipe. Must be
    10-2500 characters.
    - teaser (CharField): A short teaser/description of the recipe. May not be
    empty and must not exceed 180 characters.
    - created_on (DateTimeField): Date and time when the recipe was created. Is
    set automatically on recipe creation.
    - updated_on (DateTimeField): Date and time when the recipe was last
    updated, set automatically when recipe is updated.
    - category (IntegerField): Category of the recipe (e.g., Chicken, Pork).
    Default is 0 - "No category selected".
    - status (IntegerField): Status of the recipe (e.g., Draft, Published).
```

Settings:

Results:

All clear, no errors found

Views

Files in the *recipe_book/views directory*

recipe_book/views/__init__.py

code institute

CI Python Linter

```
1 from .recipe_list import *
2 from .features_list import *
3 from .recipe_detail import *
4 from .favourites_list import *
5 from .favourites_crud import *
6 from .ratings_crud import *
7 from .errors import *
8
```

Settings:

Results:

All clear, no errors found

recipe_book/views/favourites_crud.py

code institute

CI Python Linter

```python
import json
from django.http import JsonResponse
from django.views.decorators.http import require_POST
from ..models import Recipe, Favourite


@require_POST
def add_remove_favourite(request):
    """
    View to handle POST request to add/remove recipe from a user's favourites.
    If the user is authenticated, the recipe will be added to or removed from
    the users favourites based on whether it was already a favourite or not.
    Messages are returned to provide user feedback.

    Args:
        request (HttpRequest): HTTP request object containing the recipe id.

    Returns:
        JsonResponse: JSON response indicating success or failure.
    """
    if request.method == 'POST':
        user = request.user
        if user.is_authenticated:
            user_id = user.id
            try:
                data = json.loads(request.body)
                recipe_id = data.get("recipeId")
                is_favourite = Favourite.is_recipe_favourite_by_ids(
                    user_id, recipe_id)
                # handle case where recipe is a favourite of the user
                if (is_favourite):
```

Settings:

Results:

All clear, no errors found

recipe_book/views/favourites_list.py

code institute

CI Python Linter

```python
from django.db import models
from django.views.generic import ListView
from ..models import Recipe, Favourite


class FavouritesList(ListView):
    """
    View to display a list of a user's favorite recipes. If the user is
    authenticated, the view will return the user's favourite recipes, + some
    additional data such as average rating and rating count. If the user is not
    authenticated, the view will return an empty queryset.

    Attributes:
        model (Model): The model associated with the view (Recipe).
        template_name (str): The name of the template used for rendering the
        view.
        paginate_by (int): The number of items to paginate by.

    Methods:
        get_queryset(self): Retrieves the queryset of favorite recipes for the
        authenticated user or an empty queryset if user is not authenticated.
        Annotates each recipe object with additional details.
        get_context_data(self, **kwargs): Adds extra context data, the range
        used for creating star buttons.
    """
    model = Recipe
    template_name = "recipe_book/favourites.html"
    paginate_by = 8

    def get_queryset(self):
        """
```

Settings:

Results:

All clear, no errors found

recipe_book/views/features_list.py

code institute

CI Python Linter

```python
from django.views.generic import ListView
from django.db import models
from ..models import Recipe, Favourite


class FeaturesListView(ListView):
    """
    View to display featured recipes (highest rated and latest published).
    Annotates the recipe objects with additional details and passes additional
    details as context.

    Attributes:
        model (Model): The model associated with the view (Recipe).
        template_name (str): The template used for rendering the view.
        context_object_name (str): The name of the context object used in the
        template.

    Methods:
        get_context_data(self, **kwargs): Adds extra context data including
        highest rated recipes, latest published recipes, average rating,
        rating count, user's favorite recipes, and the range used for creating
        star buttons.
    """
    model = Recipe
    template_name = 'recipe_book/index.html'
    context_object_name = 'context'

    def get_context_data(self, **kwargs):
        """
        Adds extra context data including highest rated recipes, latest
        published recipes, the user's favourite recipes and the range user for
```

Settings:

Results:

All clear, no errors found

recipe_book/views/errors.py

code institute

CI Python Linter

```python
from django.http import HttpResponseNotFound
from django.shortcuts import render


def page_not_found_view(request, exception):
    """
    View to render a custom 404 page.

    Args:
        request (HttpRequest): The HTTP request object.
        exception (Exception): The exception that triggered the 404 error.

    Returns:
        HttpResponseNotFound: HTTP response with the rendered 404 page.
    """
    return HttpResponseNotFound(render(request, "errors/404.html", {}))


def server_error_view(request):
    """
    View to render a custom 500 page.

    Args:
        request (HttpRequest): The HTTP request object.

    Returns:
        HttpResponseNotFound: HTTP response with the rendered 500 page.
    """
    return HttpResponseNotFound(render(request, "errors/500.html", {}))

```

Settings:

Results:

All clear, no errors found

recipe_book/views/ratings_crud.py

code institute

CI Python Linter

```python
import json
from django.http import JsonResponse
from django.views.decorators.http import require_POST, require_http_methods
from ..models import Recipe, Rating


@require_POST
def add_update_rating(request):
    """
    View to handle POST request for adding or updating ratings. It expects JSON
    data in the request body with the recipe id and rating value. If the user
    is authenticated, it adds or updates the rating. It returns a JSON response
    with a message for user feedback.

    Args:
        request (HttpRequest): The HTTP request object.

    Returns:
        JsonResponse: JSON response containing the status code and a message.
    """
    if request.method == 'POST':
        # Get user
        user = request.user
        if user.is_authenticated:
            try:
                data = json.loads(request.body)
                recipe_id = data.get("recipeId")
                rating_value = data.get("rating")

            except json.JSONDecodeError:
                return JsonResponse(
```

Settings:

Results:

All clear, no errors found

recipe_book/views/recipe_detail.py

code institute

CI Python Linter

```python
import json
from django.views.generic import DetailView
from django.http import JsonResponse
from ..models import Recipe, Favourite, Rating
from ..forms import CommentForm


class RecipeDetailView(DetailView):
    """
    View for displaying details of a single recipe.

    Attributes:
        queryset (QuerySet): The queryset used to retrieve recipe objects with
        a status of published.
        template_name (str): The name of the template used to render the recipe
        detail page.
        context_object_name (str): The key used to access the recipe object in
        the template context.
        slug_url_kwarg (str): The name of the URL keyword argument containing
        the recipe slug.
    """
    queryset = Recipe.objects.filter(status=1)
    template_name = "recipe_book/recipe-page.html"
    context_object_name = "recipe"
    slug_url_kwarg = "slug"

    def get_context_data(self, **kwargs):
        """
        Adds extra context data for rendering the recipe detail template.

        Returns:
```

Settings:

Results:

All clear, no errors found

recipe_book/views/recipe_list.py

code institute

CI Python Linter

```python
from django.db import models
from django.views.generic import ListView
from django.db.models import Q
from ..models import Recipe, Favourite


class RecipeListView(ListView):
    """
    View for displaying a list of recipes based on search queries and
    categories, or all published recipes if there is no search query.

    Attributes:
        model (Model): The model associated with the view (Recipe).
        template_name (str): The name of the template used to render the page.
        paginate_by (int): The number of items to paginate by.
    """
    model = Recipe
    template_name = 'recipe_book/recipes.html'
    paginate_by = 8

    def get_queryset(self):
        """
        Retrieves the queryset of recipes based on search queries and
        categories, or all published recipes if there is no query. Sorts the
        recipes if a sort parameter is present. Annotates each recipe with
        additional details avg rating, rating count, and if user is
        authenticated, the users rating of the recipe.

        Returns:
            Queryset: A filtered queryset of Recipe objects.
        """
```

Settings:

Results:

All clear, no errors found

Tests

Files in the *recipe_book/tests directory*

recipe_book/tests/test_add_remove_favourite.py

code institute

CI Python Linter

```python
from django.test import TestCase
from django.contrib.auth.models import User
from recipe_book.models import Recipe, Favourite


class AddRemoveFavouriteViewTests(TestCase):
    """
    A test case class to test the add-remove-favourite view
    (favourites_crud.py).

    Contains tests to ensure the correct behavior of the add-remove-favourite
    function view for adding and removing favorites by authenticated users.

    Test methods:
        - `setUp`: Set up mock data for the tests.
        - `test_add_favourite_authenticated_user`: Test that a favourite is
        added when the request is by a valid user and recipe, and favourite
        does not already exist.
        - `test_remove_favourite_authenticated_user`: Test that a favourite is
        removed when the request is by  valid user and recipe, and favourite
        already exists.
    """
    def setUp(self):
        """ Set up mock data for testing """
        self.super_user = User.objects.create_superuser(
            username="testsuperuser",
            email="testsuper@test.com",
            password="supertestpassword"
        )

        self.user = User.objects.create_user(
```

Settings:

Results:

All clear, no errors found

recipe_book/tests/test_add_update_rating_view.py

code institute

CI Python Linter

```python
from django.contrib.auth.models import User
from django.test import TestCase
from recipe_book.models import Recipe, Rating


class AddUpdateRatingViewTest(TestCase):
    """
    A test case class to test the add-update-rating view, in ratings_crud.py.

    Contains tests to ensure the correct behavior of the add-update-rating
    view.

    Test methods:
        - `setUp`: Set up mock data for the tests.
        - `test_add_rating_authenticated_user`: Test method is returning
        correct response, 200, and rating is created, when request with
        authenticated user and no existing rating for recipe.
        - `test_update_rating_authenticated_user`: Test method is returning the
        correct response, 200, and rating is updated with new rating value,
        when request with an authenticated user and an existing rating.
    """
    def setUp(self):
        """ Set up mock data for testing """
        self.super_user = User.objects.create_superuser(
            username="testsuperuser",
            email="testsuper@test.com",
            password="supertestpassword"
        )

        self.user = User.objects.create_user(
            username='testuser', password='testpassword')
```

Settings:

Results:

All clear, no errors found

recipe_book/tests/test_delete_rating_view.py

code institute

CI Python Linter

```python
from django.test import TestCase
from django.contrib.auth.models import User, AnonymousUser
from django.http import JsonResponse
from recipe_book.views import delete_rating
from recipe_book.models import Rating, Recipe
from unittest.mock import Mock


class DeleteRatingViewTest(TestCase):
    """
    A test case class to test the delete_rating view in ratings_crud.py.

    Contains tests to ensure the correct behavior of the delete_rating view
    for both authenticated and unauthenticated users, and scenarios where
    the rating exists or does not exist.

    Test methods:
        - `setUp`: Set up mock data for the tests.
        - `test_delete_rating_authenticated_user`: Test a delete request with
        an authenticated user,
        with an existing rating for recipe. Ensure getting the correct
        response and that the rating is deleted.
        - `test_delete_rating_not_authenticated_user`: Test a delete request
        with an anonymous user. Ensure getting the correct response, 401, and
        that the rating is not deleted.
        - `test_delete_rating_does_not_exist`: Test a delete request with a
        logged in user, but the rating does not exist. Ensure getting the
        correct response, 400.
    """
    def setUp(self):
        """ Set up mock data for testing """
```

Settings:

Results:

All clear, no errors found

recipe_book/tests/test_favourites_list_view.py

code institute

CI Python Linter

```python
from django.contrib.auth.models import User
from django.test import TestCase
from django.urls import reverse
from recipe_book.models import Recipe, Favourite


class TestFavouritesListView(TestCase):
    """
    A test case class to test the FavouritesListView.

    Contains tests to ensure the correct rendering of the favourites list for
    both authenticated and unauthenticated users.

    Test methods:
        - `setUp`: Set up mock data for the tests.
        - `test_favourites_page_status_code`: Test the status code of the
        favourites list page.
        - `test_favourites_page_template_used`: Test the correct template is
        used for the FavouritesView.
        - `test_authenticated_user_view`: Test that the favourites page
        displays the favourited recipes of a logged in user.
        - `test_unauthenticated_user_view`: Test that the favourites page does
        not contain recipe objects when the user is unauthenticated.
    """
    def setUp(self):
        """ Set up mock data for testing """
        self.user = User.objects.create_user(
            username="testuser", email="test@test.com", password="testpassword"
        )
        self.client.login(username="testuser", password="testpassword")

```

Settings:

Results:

All clear, no errors found

recipe_book/tests/test_features_list_view.py

code institute

CI Python Linter

```python
from django.test import TestCase
from django.urls import reverse


class TestFeaturesListView(TestCase):
    """
    A test case class to test the FeaturesListView.

    Contains tests to ensure the correct rendering of the features list page.

    Test methods:
        - `test_features_list_page_status_code`: Test the status code of the
        retrieving the features list page (home page).
        - `test_features_list_page_template_used`: Test the correct template is
        used for the FeaturesListView.
    """

    def test_features_list_page_status_code(self):
        """ Test the status code of the home page. """
        response = self.client.get(reverse('home_page'))
        self.assertEqual(response.status_code, 200, msg="Status code not 200")

    def test_features_list_page_template_used(self):
        """ Test the correct template is used for the FeaturesListView. """
        response = self.client.get(reverse('home_page'))
        self.assertTemplateUsed(response, 'recipe_book/index.html')

```

Settings:

Results:

All clear, no errors found

recipe_book/tests/test_models_favourite.py

code institute

CI Python Linter

```python
from django.contrib.auth.models import User
from django.test import TestCase
from django.db import IntegrityError
from recipe_book.models import Recipe, Favourite


class TestFavouriteModel(TestCase):
    """
    A test case class to test the functionality of the Favourite model.

    Contains tests to ensure the correct creation, deletion, and retrieval
    of Favourite instances.

    Test methods:
        - `setUp`: Creates mock data testing.
        - `test_favourite_creation`: Test creating a Favourite given a
        logged-in user and a recipe.
        - `test_unique_favourite_constraint`: Test to ensure the same user
        cannot favourite the same recipe twice.
        - `test_favourite_deletion`: Test that a Favourite object is deleted as
        intended.
        - `test_is_recipe_favourite`: Test to ensure the
        `is_recipe_favourite()` method returns True when the given recipe is a
        favourite of the given user, and False when not.
        - `test_is_recipe_favourite_by_ids`: Test to ensure the method returns
        True when passed a recipeId and userId when the recipe is a favourite
        of the user, and False when not.
        - `test_get_user_favourite_ids`: Test to check that
        `get_user_favourite_ids` returns the right recipe ids given the user
        has favourited two recipes.
        - `test_create_favourite`: Test if the method returns True when
```

Settings:

Results:

All clear, no errors found

recipe_book/tests/test_models_rating.py

code institute

CI Python Linter

```python
from django.contrib.auth.models import User
from django.test import TestCase
from django.db import IntegrityError
from recipe_book.models import Recipe, Rating


class TestRatingModel(TestCase):
    """
    A test case class to test the functionality of the Rating model.

    Contains tests to ensure the correct creation, deletion, and getting of
    Rating instances.

    Test methods:
        - `setUp`: Creates mock data for testing.
        - `test_rating_creation`: Test creating a rating given a logged-in user
        and a recipe.
        - `test_unique_rating_constraint`: Test to ensure the same user cannot
        rate the same recipe twice.
        - `test_rating_deletion`: Test that a Rating object is deleted as
        intended.
        - `test_get_recipe_avg_rating_with_ratings`: Test to ensure the method
        returns the average rating for a given recipe with ratings.
        - `test_get_recipe_avg_rating_no_ratings`: Test to ensure the method
        returns 0.0 when a given recipe has no ratings.
        - `test_get_recipe_no_of_ratings`: Test that the method returns the
        number of ratings for a given recipe.
        - `test_get_user_rating_of_recipe`: Test that the method returns the
        user's existing rating value for a given recipe, or None if the user
        has not rated the recipe.
    """
```

Settings:

Results:

All clear, no errors found

recipe_book/tests/test_models_recipe.py

code institute

CI Python Linter

```python
from django.contrib.auth.models import User
from django.test import TestCase
from django.utils import timezone
from django.db import IntegrityError
from django.core.exceptions import ValidationError
from datetime import datetime
from recipe_book.models import Recipe


class TestRecipeModel(TestCase):
    """
    A test case class to test the functionality of the Recipe model.

    Contains tests to ensure the correct creation and validation of Recipe
    instances.

    Test methods:
        - `setUp`: Creates a mock superuser and Recipe instance for testing.
        - `test_recipe_creation`: Test the creation of a Recipe instance.
        - `test_unique_title`: Test that a Recipe with a duplicate title
        cannot be created.
        - `test_unique_slug`: Test that a Recipe with a duplicate slug cannot
        be created.
        - `test_title_max_length`: Test that a validation error is raised when
        the title length exceeds the maximum limit.
        - `test_slug_max_length`: Test that a validation error is raised when
        the slug length exceeds the maximum limit.
        - `test_alt_text_max_length`: Test that a validation error is raised
        when the alt text length exceeds the maximum limit.
        - `test_teaser_max_length`: Test that a validation error is raised when
        the teaser length exceeds the maximum limit.
```

Settings:

Results:

All clear, no errors found

recipe_book/tests/test_recipe_detail_view.py

code institute

CI Python Linter

```python
import json
from django.contrib.auth.models import User
from django.test import TestCase
from django.urls import reverse
from recipe_book.models import Recipe, Favourite, Comment


class TestRecipeDetailView(TestCase):
    """
    A test case class to test the functionality of RecipeDetailView.

    Contains tests to ensure the correct rendering of the recipe detail page,
    including rendering with logged in vs not logged in user, posting comments,
    editing comments, and deleting comments.

    Test methods:
        - `setUp`: Sets up test mock data.
        - `test_render_recipe_detail_page_with_invalid_slug`: Test rendering a
        recipe detail page with an invalid slug.
        - `test_render_recipe_detail_page_when_not_logged_in`: Test rendering a
        recipe detail page when the user is not logged in.
        - `test_render_recipe_detail_page_with_favourite`: Test rendering a
        recipe detail page when the recipe is favorited by the logged-in user.
        - `test_render_recipe_detail_page_with_nonfavourited_recipe`: Test
        rendering a recipe detail page when the recipe is not favorited by the
        logged-in user.
        - `test_post_valid_comment`: Test posting a valid comment.
        - `test_post_comment_not_authenticated`: Test posting a comment while
        not logged in.
        - `test_post_invalid_comment`: Test posting an invalid comment.
        - `test_put_valid_edit_of_comment`: Test making a valid edit of a
```

Settings:

Results:

All clear, no errors found

recipe_book/tests/test_recipe_list_view.py

code institute

CI Python Linter

```python
from django.contrib.auth.models import User
from django.test import TestCase
from django.urls import reverse
from recipe_book.models import Recipe, Favourite


class TestRecipeListView(TestCase):
    """
    A test case class to test the functionality of RecipeListView.

    Contains tests to ensure the correct rendering of the recipe list page,
    including pagination, search functionality, and category filtering.

    Test methods:
        - `setUp`: Sets up test mock data.
        - `test_render_recipe_list_page`: Test rendering the recipe list page.
        - `test_render_recipe_list_page_with_search_query`: Test rendering the
        recipe list page with a search query.
        - `test_render_recipe_list_page_with_no_result`: Test rendering the
        recipe list page with a search query for which there is no matching
        result.
        - `test_render_recipe_list_page_with_category_search`: Test rendering
        the recipe list page with a category search result.
        - `test_recipe_list_page_pagination`: Test pagination on the recipe
        list page.
        - `test_recipe_list_page_not_paginated`: Test to ensure the recipe
        list page is not paginated when the number of recipes is smaller than
        the paginate_by number.
    """

    def setUp(self):
```

Settings:

Results:

All clear, no errors found