Django EmailBackend Explained

Line-by-Line Explanation

1. **from django.contrib.auth.backends import ModelBackend**
- Imports Django's built-in authentication backend, which provides the default methods for authentication and permissions.

2. **from django.contrib.auth import get_user_model**
- Imports a function that retrieves the active user model in the project. This is useful because you're using a custom user model (`CustomUser`).

3. **class EmailBackend(ModelBackend):**
- Defines a new backend class called *EmailBackend* that extends Django's `ModelBackend`.

4. **def authenticate(self, request, username=None, password=None, **kwargs):**
- Overrides the `authenticate` method. This is the method Django calls whenever a user tries to log in.
- The `username` parameter here will actually contain the email address (because we customized the login form).

5. **UserModel = get_user_model()**
- Fetches the current user model (`CustomUser` in your case). This makes the backend flexible in case the user model changes.

6. **try: user = UserModel.objects.get(email=username)**
- Tries to fetch a user where the *email* field matches the value the user entered in the login form.
- Example: If the user types "john@example.com", Django checks `CustomUser.objects.get(email="john@example.com")`.

7. **except UserModel.DoesNotExist:**
- If no user exists with that email, return `None` (authentication fails).

8. **else: if user.check_password(password): return user**
- If the user was found, checks whether the entered password matches the hashed password stored in the database.
- If the password is correct, return the `user` object.

9. **return None**
- If no matching user or incorrect password, authentication fails.

Example of How It Works

Suppose a user tries to log in with:
- Email: *alice@example.com*
- Password: *mypassword*

Here's what happens step by step:
1. Django calls `authenticate(request, username="alice@example.com", password="mypassword")`.
2. Inside the backend, `UserModel = get_user_model()` → this points to your `CustomUser` model.
3. It tries: `user = CustomUser.objects.get(email="alice@example.com")`.
4. If such a user exists, it checks: `user.check_password("mypassword")`.

5. If the password matches, it returns the `user` object → Django logs the user in.
6. If the email doesn’t exist or the password is wrong, it returns `None` → login fails.

This way, your login system uses **email + password** instead of **username + password**.