Understanding Django Email Backend Authentication

In Django, the authentication backend can be customized so users log in with their email instead of username.

Code snippet:

```
--------------------------------------------------
from django.contrib.auth.backends import ModelBackend
from django.contrib.auth import get_user_model

class EmailBackend(ModelBackend):
def authenticate(self, request, username=None, password=None, **kwargs):
UserModel = get_user_model()
try:
user = UserModel.objects.get(email=username)
except UserModel.DoesNotExist:
return None
else:
if user.check_password(password):
return user
return None
```

Explanation line by line:

----

1. from django.contrib.auth.backends import ModelBackend
- Imports Django's default authentication system that we are extending.

2. from django.contrib.auth import get_user_model
- Dynamically retrieves the active User model (e.g., CustomUser).

3. class EmailBackend(ModelBackend):
- Creates our custom backend that overrides authenticate().

4. def authenticate(..., username=None, password=None, **kwargs):
- Django always passes the login identifier into the 'username' argument,
even if we call it "Email" on the form.

5. UserModel = get_user_model()
- Stores your User model so queries can be made against it.

6. user = UserModel.objects.get(email=username)
- Tries to fetch the user whose email field matches what the user typed into the login form.
- Even though the variable is called 'username', we are treating it as an email here.

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